volume 3

Disassembled Handbook for TRS-80

Richcraft Engineering Ltd.
Drawer 1065
Chautauqua, New York 14722

FIRST PRINTING 1981

VOLUME III

DISASSEMBLED HANDBOOK FOR TRS-80

Robert M. Richardson

Published by: Richcraft Engineering Ltd.
Drawer 1065, Wahmeda Industrial Park
Chautauqua, New York 14722
(716) 753-2654

VOLUME 1:
1st printing - January 1980
2nd printing - February 1980
3rd printing - April 1980
4th printing - June 1980
5th printing - October 1980
6th printing - December 1980 (German ed.)
7th printing - December 1980 (French ed.)

VOLUME 2:
1st printing - May 1980
2nd printing - July 1980
3rd printing - October 1980
4th printing - December 1980 (German ed.)
5th printing - December 1980 (French ed.)

VOLUME 3:
1st printing - January 1981
2nd printing - January 1981 (German ed.)
3rd printing - January 1981 (French ed.)

German and French language editions available on special order from Richcraft Engineering Ltd.

- VOLUME III -

CHAPTER	CONTENTS	PAGE

NOTICE:

 All errors and sins of omission are unintentional. Special thanks are due the National Semiconductor Corporation and Telesis Laboratory for the use of their drawings in Chapters 6 and 7, plus ALL the MANY other contributors who made this Volume possible.

- FOREWORD -

A great many people and firms have indirectly contributed to to Volume 3. The most prominent is:

Dr. Federico Faggin

Dr. Federico Faggin, now Group Vice President of Exxon Enterprises with responsibility for the newly formed Computer Systems Group.

He virtually, single-handedly, created the microprocessor and the revolution it has bought about. He was leader of the team at Intel that developed the 4004, 4040, 8008 and 8080 microprocessors.

He was founder and President of Zilog that developed the Z-80 which captured the lion's share of the worldwide advanced 8 bit microprocessor market and is now leading the 16 bit microprocessor race with the new Zilog Z-8000.

EVERY microprocessor AND microcomputer in the world today, the USSR included, has Dr. Federico Faggin's genius written all over it.

This book would never have been written without Dr. Faggin's multitudinous contributions to the science, the technology, and the art of microprocessing.

Thank you, spirit of Charles Tandy for making the TRS-80 a happening.

All of us Fagginsylvanians appreciate BOTH of your contributions to making our world a better place to live.

Robert M. Richardson
Chautauqua Lake, New York

- INTRODUCTION -

One of the easiest and most fascinating ways to become familiar with the Z-80 instruction set for assembly language programming is to write your own disassembler. In Chapter 1 we will do just that. If you will take the time to actually sit down and ENTER and RUN the program, you will find that all of a sudden the beauty and logic of the Z-80 instruction set will permeate your grey matter and it will miraculously become fixed in place; i.e., you will remember it till time immemorial. As you become intimately familiar with disassemblers and how they operate, you will probably ask yourself, as we did, how to speed up the disassembly process? From little thoughts like that, big contests grow. The 'Great BASIC Disassembler Fastest Footrace Contest' was announced in a number of publications during May and June 1980 with a closing date of September 30, 1980. Entries poured in from the rock bound coasts of New England to the sunny shores of California.

Chapter 2 has the FIRST PLACE winning disassembler program in it that 'will blow your mind' when you time it. It is truly FAST, accurate, and will keep up with many assembly language disassemblers. It will disassemble the first 6000 bytes of Level II ROM in just a 'hair' over 2 minutes. Chapter 2 also includes the 'Most Unique' prize winning disassembler program. It won this prize for two reasons: 1) the program is one of the most beautifully written (both in looks and style) BASIC programs we have ever had the pleasure to see. 2) the program is FAST too, as in speedy. You can learn a great deal by studying BOTH of these excellent programs.

Chapter 3 presents both a BASIC and an assembly language program to illustrate SPOOLING; i.e., suppposedly doing two things at the same time. Actually, all a SPOOLER does is to use the WAITing period for slow peripherals (like a cassettee or line printer) to perform some useful function.

Chapter 4 covers ALL the Z-80 interrupts and a few mini-interrupt programs using the IM1 mode that is available to us in the TRS-80 without circuit board modifications.

Chapter 5 is a BIG Chapter that pretty well covers the water-front regarding how to interface the TRS-80 to the outside real world with homebrew modifications, if desired. IF you are not a homebrew enthusiast, the last 17 pages of this Chapter explores all the goodies available to the TRS-80 user via the Telesis VAR/80 interface.

Chapter 6 is another BIG Chapter. It features a homebrew analog to digital converter, reviews most all of the A/D types available today, plus 11 pages on the excellent Alpha Products model 'Analog 80' A/D conversion system. You will have an equivalent graduate degree on the subject when you finish this long, but interesting Chapter.

Chapter 7 hopefully covers the waterfront on the subject of digital to analog converters. Again, we start out with a home-brew variety to get a 'feel' for the problems involved with this variety of peripheral, then a brief survey of the background of D/A development and types available, and finally, build a moderately fast and accurate system using the VERY low cost DAC0808 converter with the LF351 operational amplifier.

Chapter 8's Morse TRANSMIT program was written by Charles D. House, winner of the 'Fastest Footrace Contest.' It is included to illustrate Chick's (Col. House) approach to the problem using the BASIC ONGOTO function instead of the BASIC IF-THEN function used by the author in Volume 2's Chapter 10. The limiting speed factors are lines 76 & 77 which peel off the dots and dashes providing correct timing, hence both programs run at identical speeds. It is an interesting approach.

Chapter 9 presents about the minimum "get started" communications bulletin system assembly language program. Also included, is a considerable amount of philosophizing about program swapping and copyrights. The author sees nothing wrong with exchanging programs for FREE. Just don't sell them is the message here. Also, the author forecasts program swapping via computer bulletin boards in the next few months that will drive those who object to program swapping up the proverbial wall.

Chapter 10 is the real 'Grandaddy' Chapter of Volume 3; all 41 pages of it. It is divided into 3 sections:

1. A bit of teletype (tm) history and suggestions for those who would like to obtain an amateur radio license as painlessly as possible. Putting together a 6 and 2 meter amateur station that will provide an ASCII data link over most ANY line of sight range. An assembly language program that will allow you to operate half duplex with the other station AND transmit or receive BASIC, source code, and object code programs from one station to another, PLUS store these programs anywhere in MEM for later recall and use, as you see fit.

2. A truly candid survey of the Macrotronics M80 Morse Code and Baudot Radio Teletype (tm) System including the hardware adaptor board that comes with it. Both good and bad features are covered on this early system. Hopefully, they have been recently corrected and improved.

3. The last section should be re-titled: "An ASCII RTTY Receiving System Using the Radio Shack RS-232C Interface and Telephone II MODEM." This should be of interest to those TRS-80 users who are either radio amateurs or computer buffs with access to a communications receiver that will tune the low frequency ham bands. Raise your hand and promise: "I will not peek into this section before reading ALL of the preceeding parts of Chapter 10." Radio Shack was as surprised as we were.

Chapter 11 consists of '20 questions' for each of the 10 Chapters. The questions are intentionally quite EASY, but in most cases do require a thorough understanding of the principle involved.

If you do not have the answer to each question on the tip of your tongue, go back and re-read the section involved rather than looking up the answer in the next Chapter. IF you do choose to look up the answer without re-reading and undertanding the appropriate section, the only one you are cheating is YOURSELF.

Chapter 12 has the answers to each of the 20 questions for each of the 10 Chapters. One of the most important aspects of a self-programmed teaching text is that each reader may proceed at his/her own pace. There is no peer pressure to slow down if you get ahead of the class and no one to laugh at you if you miss a question and fall behind as YOU ARE THE CLASS. Go as fast or as slow as you wish. It is entirely up to you.

We recommend reading the Chapters in the sequence presented as in most of them we presume an understanding of those facts and principles previously covered. Chapter 10 is an exception, in that we tried to write it on a stand alone basis for eventual reprinting and use by computer and/or amateur radio clubs. We ask your understanding for the few duplications from Chapter 9 that were necessary to achieve this end.

APPENDIX A: Is a combined index of the topics/subjects covered in Volumes 1, 2, and 3. It should be of some help to the reader who covers each Volume a semester or more apart. Do not be ashamed to use it, as we use it ourselves frequently. The German and French language editions contain Appendix A1 in English and Appendix A2 in their respective languages.

APPENDIX B: Is a list of most all programs from Volumes 1, 2, and 3 available on 35 track disks. The disks have been made primarily for educators use in classroom applications. As such, they may prepare their own course outlines that are totally independent of the authors' text, yet flow in the same pattern. Many teachers have chosen to do so with Volumes 1 and 2 at a number of prestigious universities. We encourage this practice and are highly complimented. All the programs are on 1 two-sided disk or 2 single-sided disks, as desired.

VOLUME 4: Will be published during the summer of 1981 at approx. $22/copy. Reservations accepted at $18 through May 1981. SEND NO DEPOSIT - NO MONEY !

- CHAPER 1 -

WRITING DISASSEMBLER PROGRAMS

INTRODUCTION:

What did he say? Ye gods, not another disassembler. What with with Apparat's NEWDOS+ and NEWDOS 80 disassemblers, Small System's many disassemblers, Gerald Abear's disassembler, and venerable Pastor George Blank's disassembler, who needs another one and why?

I am glad you asked this question, Gridley. Your questions are always pertinent and sometimes even to the point.

In Volume 2's first Chapter we listed all 694 of the Z-80 instructions in decimal numerical order (and most in hex too). Ostensibly, the purpose was to allow those readers WITHOUT disassemblers to decode the 256+ blanked out object codes and 256+ blanked out source codes for the disassembled Level II ROM in Chapter 2. The purpose for blanking out these codes was to protect Microsoft's copyrighted program and to protect the author from any legal entanglements since our counsel advised that reproducing PART of a copyrighted software program that would NOT work, was not a copyright violation.

Counsel's advice was well taken at first, but a few weeks after Volume 2 was published with the partially disassembled Level II ROM in it, gnawing specters of doubt appeared in the dark corners of the night. What if counsel was wrong? They never seem to 'hang' the counsel when he loses a case. What would happen if we were sued and the court awarded our trusty 1960 Jeep with snowplow as damages to the plaintiff? Would we starve in the winter when we could not get out? All sorts of fascinating horror stories conjured themselves up in our imagination after the sun went down.

One night I was awakened by a strange drumming sound much like a tom-tom or a horde of bats flying about. Then a bright dazzling LIGHT appeared and the spirit of the ethereal Indian (US) Princess Wahmeda, appeared before me mumbling: "WAH - MEE - DAH will let no harm come to the humble paleface who lives at the site of her former teepee. Follow my instructions to the letter and only good fortune will come to you. Write a BASIC disassembler that is faster and fleeter than the other BASIC competition and all will be well." The light slowly faded and the spirit of the Princess vanished. I was dumbfounded for a while, and then rolled over and went back to sleep.

Jumping barefoot out of bed the next morning into a great heaping pile of bat dung, I recalled the strange dream, or apparition, of the previous night. Had the Princess really left a calling card to remind me of her visit and advice?

Not being in the least bit inclined toward superstition, I nevertheless NEVER walk under ladders and NEVER walk between aircraft propeller blades even if the engine is positively OFF. Meanwhile, the canker gnawed till I finally decided that regardless of the spirit, dream, manifestation of bad booze, or whatever, I had best sit myself down and write as FLEET (as in fast) a BASIC disassembler as was possible. The canker thereupon immediately disappeared. No more barefoot bat dung in the morning. I slept well and soundly. The sprite, spirit or whatever it was, was completely exorcised.

Now that you know the sound-scientific-entirely-rational and logical reasons for this Chapter, let's get on with it.

If perchance you dream and/or count your fingers and toes in hexadecimal, AND have memorized all 694 Z-80 instructions absolutely cold in both decimal and hex, MAYBE this Chapter is not for you. GOTO Chapter 2 though you do not get to pass GO nor collect $200.

HOW ABOUT ALL THOSE OTHER DISASSEMBLERS WRITTEN IN BASIC ? ? ?

Most disassemblers written in BASIC rather than assembly language are SLOW as the proverbial tortoise on a very hot day. Whyfore this lackadaisical pace? Quite simple, Gridley. They were written using BASIC's READ-DATA functions because the authors were either too lazy (not a criticism as this author is undoubtedly the laziest person he knows) to consider other methods, or they did not care about speed (though 2 days to disassemble ROM does seem a bit much), or they were trying to preserve precious memory (which is so cheap now, as to be almost free). No matter....no nevermind. We will write 2 similar disassemblers....and we will do it together. Do not forget that "magic" that transpires when YOU sit down at the 'ole TRS-80 keyboard and ENTER and RUN a program yourself. You will understand and retain in your own personal MEM the points we cover a zillion times better than IF you merely read the text and skip onto the next Chapter. Sad, but true. After all, sitting down at your keyboard & ENTERing 694 lines of object and sources codes is no fun at all. But believe me, YOU will remember the Z-80 instructions and important points when you are finished and definitely will be a much more skilled assembly language programmer for the extra effort.

HOW WE GONNA BEAT MR. ABEAR AND PREACHER BLANK IN TODAY'S GREAT TRS-80 FLEET DISASSEMBLER FASTEST FOOT RACE ? ? ? ?

No, Gridley, you cannot cheat and POKE a machine language program into MEM via READ-DATA statements....I know YOU and the rules and will be watching closely with my referee's whistle IF you try to pull any of that kid stuff. The program MUST be written in BASIC. Of course, you may use PEEK and POKE somewhat, but the VAST majority of the program MUST be written in BASIC. Here's a hint. Before you turn the page, GOTO Vol. 2's pages 150 & 151 and see how we obtain 35 words per minute transmit MORSE code speed from a simple BASIC program.

Let us briefly retrogress for a page or so back into the world of Level II BASIC. With a title like ours we feel free to disassemble most anything in sight. Since Volumes 1 & 2 did not include the decimal codes used in BASIC programs to identify the BASIC functions, why not have a go at disassembling these codes with the following sophisticated 3 liners. The first BASIC program is for DOS 2.1, DOS 2.2, DOS 2.3, NEWDOS+ or NEWDOS 80. The second BASIC program is for non-disk TRS-80s. Actually, they are not included just as a FILLER for this page. We will be using one or more of these BASIC instruction codes later in this Chapter to play some weird tricks with the second disassembler program.

DISK PROGRAM:

```
10 REM = THE BASIC FUNCTION FOR 'X'
20 INPUTX:POKE26814,X:CLS:PRINT"        X=";X:PRINT:LIST10
30 POKE26814,147:GOTO10
```

NON-DISK PROGRAM:

```
10 REM = THE BASIC FUNCTION FOR 'X'
20 INPUTX:POKE17133,X:CLS:PRINT"        X=";X:PRINT:LIST10
30 POKE17133,147:GOTO10
```

As you may easily see, these programs are as sophisticated as apple jelly and peanut butter sandwiches. Will they decode ALL the BASIC functions BASIC codes? Sure they will Gridley. Give it a try. You know that the decimal BASIC codes from 8 to 127 contain the control codes and ASCII character codes as listed on pages C/1 and C/2 of the "Level II Basic Reference Manual." What we are going to do is let the above simple BASIC programs disassemble the decimal BASIC instruction codes for us. It will take you only about 5 minutes to accomplish the task. Load the appropriate program and RUN. Start off with decimal '128' input and then ENTER. Sonofagun, it works.

PROGRAM OUTPUT:

```
        X = 128

       END = THE BASIC FUNCTION FOR 'X'
```

To obtain the next BASIC function, type RUN30 and input 129, ENTER. Obviously what the program is doing is POKEing the function code into MEM where REM used to be and then very conveniently printing out the code's function on the video display via the LIST10 statement. After your first RUN, the RUN30 merely POKEs 147 (= REM) back into MEM to avoid an illegal function call, error, from BASIC. Go ahead and try RUN30, input 129 and ENTER. The Level II BASIC codes for BASIC programs are listed on the next page.

```
        X = 129

       FOR = THE BASIC FUNCTION FOR 'X'
```

- LEVEL II FUNCTION CODES FOR PROGRAMS WRITTEN IN BASIC -

FUNCT	DEC	FUNCT	DEC	FUNCT	DEC	FUNCT	DEC
ABS	217	AND	210	ASC	246	ATN	228
AUTO	183	CDBL	241	CHR$	247	CINT	239
CLEAR	184	CLOAD	185	CLOSE	166	CLS	132
CMD	133	CONT	179	COS	225	CSAVE	186
CSNG	240	CVD	232	CVI	230	CVS	231
DATA	136	DEF	176	DEFDBL	155	DEFINT	153
DEFSNG	154	DEFSTR	152	DELETE	182	DIM	138
EDIT	157	ELSE	149	END	128	EOF	233
ERL	194	ERR	195	ERROR	158	EXP	224
FIELD	163	FIX	242	FN	190	FOR	129
FRE	218	GET	164	GOSUB	145	GOTO	141
IF	143	INKEY$	201	INP	219	INPUT	137
INSTR	197	INT	216	KILL	170	LEFT$	248
LEN	243	LET	140	LINE	156	LIST	180
LLIST	181	LOAD	167	LOC	234	LOF	235
LOG	223	LPRINT	175	LSET	171	MEM	200
MERGE	168	MID$	250	MKD$	238	MKI$	236
MKS$	237	NAME	169	NEW	187	NEXT	135
NOT	203	ON	161	OPEN	162	OR	211
OUT	160	PEEK	229	POINT	198	POKE	177
POS	220	PRINT	178	PUT	165	RANDOM	134
READ	139	REM	147	RESET	130	RESTORE	144
RESUME	159	RETURN	146	RIGHT$	249	RND	222
RESET	172	RUN	142	SAVE	173	SET	131
SGN	215	SIN	226	SQR	221	STEP	204
STOP	148	STR$	244	STRING$	196	SYSTEM	174
TAB (	188	TAN	227	THEN	202	TIME$	199
TO	189	TROFF	151	TRON	150	USING	191
USR	193	VAL	245	VARPTR	192	+	205
-	206	/	208	*	207	>	212
<	214	↑	209	&H	38/72	'	NOTE
RSET	172	=	213				

NOTES:

1) Apostrophe ' = REM abbreviation has 3 nos. 58 - 147 - 251.

2) Decimal 251, 252, 253, 254 & 255 are control codes. Their function is NOT as shown below, but 3) to 7) illustrate their action in this mini - 3 line program.

3) 251 = move to right edge of video display.

4) 252 = wipe out all to right of line number.

5) 253 = wipe line number.

6) 254 = replace line number with '85'.

7) 255 = replace line number with 'ISA'.

BACK TO WRITING A FLEET DISASSEMBLER PROGRAM IN BASIC:

Ok Gridley, did you notice ANY variety of READ-DATA statements in Volume 2's Morse Code TRANSMIT program on pages 150 & 151?

PLEASE wake him up.

No, you did not. READ-DATA statements are about the "slowest" and most time consuming of all the BASIC functions/statements we have to choose from in Level II for this application. We wrote a number of Morse transmit programs using READ-DATA in most every format conceivable and the highest MAX code speed obtainable was in the 5 to 10 words per minute ballpark. The IF-THEN statements are at least 6 to 8 times faster, and even more so IF we add the GOTO address after each IF-THEN.

Another sneaky "speed" trick is to divide down the number of IF-THEN statements into segments of about 32 addresses and route the program to the applicable beginning address of the appropriate segment.

What did he say? If he's so smart why didn't he use it in the Vol. 2 Morse program?

Another good question, Gridley. Glad you are back with us. The reason we did not use it in the Morse program was that had we done so, the transmit program would have run FASTER than the little TRS-80 cassette motor control relay could follow. You will remember that we used this relay to key either a TTL chip or 5 Volt dc high speed reed type keying relay to key the transmitter or code practice oscillator. Since the speed of the cassette relay was the limiting factor, there was no point in increasing program speed. We actually hacked our way into the keyboard and installed a jack on the rear that allowed us plug-in a TTL chip (a 7407) between the cassette relay and Z-41, the TRS-80 relay driver chip. This mod allowed us to increase program speed to about 50 to 60 words per minute. This was never published or implemented for two reasons: very few hams can copy Morse much beyond the 20 word per minute level, and, secondly, we discourage inexperienced hackers from getting inside the TRS-80 with a hot soldering iron as it often is an invitation to a very expensive disaster. Better they should learn to fly helicopters. It would be much safer.

Our first disassembler is (we believe) the simplest and fastest disassembler that can be written in BASIC without cheating or breaking the rules of the game. It is certainly NOT the shortest (minimum MEM) disassembler that can be written, but we still maintain that its claim to fame is:

1) speed 2) simplicitiy 3) understandability

Like the flag and motherhood, it has its virtues. Let's look at it now. After the umpteen pages of program (nobody told you it was a SHORT disassembler), we'll analyze its operation.

```
10 ' A TRULY WEIRD  B A S I C  DISASSEMBLER BY W4UCH/2 - WEIRD
15 '
20 ' COPYRIGHT (C) 1980 BY RICHCRAFT ENGINEERING LTD.
25 '
30 DEFINTA-C,E-Z:CMD"T"
35 INPUT"MEM BEGINNING , ENDING IN DECIMAL";A,B:CLS
40 PRINT"MEMORY","OBJECT","Z-80","DATA OR"
45 PRINT"ADDRESS","CODE","INSTRUCT.","ADDRESS"
50 IFA=0THENPRINT" 0"," 243","DI (DIS INT)":A=A+1
55 IFA>BGOTO55
60 C=PEEK(A)
65 IFC<32GOTO100ELSEIFC<64GOTO132ELSEIFC<96GOTO640ELSEIFC<128GOT
O960ELSEIFC<160GOTO1280ELSEIFC<192GOTO1600ELSEIFC<224GOTO1920
70 GOTO2240
100 IFC=0THENA$="NOP":GOTO3010
101 IFC=1THENA$="LD BC,DATA":GOTO3030
102 IFC=2THENA$="LD (BC),A":GOTO3010
103 IFC=3THENA$="INC BC":GOTO3010
104 IFC=4THENA$="INC B":GOTO3010
105 IFC=5THENA$="DEC B":GOTO3010
106 IFC=6THENA$="LD B,DATA":GOTO3020
107 IFC=7THENA$="RLCA":GOTO3010
108 IFC=8THENA$="EX AF,AF'":GOTO3010
109 IFC=9THENA$="ADD HL,BC":GOTO3010
110 IFC=10THENA$="LD A,(BC)":GOTO3010
111 IFC=11THENA$="DEC BC":GOTO3010
112 IFC=12THENA$="INC C":GOTO3010
113 IFC=13THENA$="DEC C":GOTO3010
114 IFC=14THENA$="LD C,DATA":GOTO3020
115 IFC=15THENA$="RRCA":GOTO3010
116 IFC=16THENA$="DJNZ":GOTO3050
117 IFC=17THENA$="LD DE,DATA":GOTO3030
118 IFC=18THENA$="LD (DE),A":GOTO3010
119 IFC=19THENA$="INC DE":GOTO3010
120 IFC=20THENA$="INC D":GOTO3010
121 IFC=21THENA$="DEC D":GOTO3010
122 IFC=22THENA$="LD D,DATA":GOTO3020
123 IFC=23THENA$="RLA":GOTO3010
124 IFC=24THENA$="JR ADDR":GOTO3050
125 IFC=25THENA$="ADD HL,DE":GOTO3010
126 IFC=26THENA$="LD A,(DE)":GOTO3010
127 IFC=27THENA$="DEC DE":GOTO3010
128 IFC=28THENA$="INC E":GOTO3010
129 IFC=29THENA$="DEC E":GOTO3010
130 IFC=30THENA$="LD E,DATA":GOTO3020
131 IFC=31THENA$="RRA":GOTO3010
132 IFC=32THENA$="JR NZ,ADDR":GOTO3050
133 IFC=33THENA$="LD HL,DATA":GOTO3030
134 IFC=34THENA$="LD (ADDR),HL":GOTO3030
135 IFC=35THENA$="INC HL":GOTO3010
136 IFC=36THENA$="INC H":GOTO3010
137 IFC=37THENA$="DEC H":GOTO3010
138 IFC=38THENA$="LD H,DATA":GOTO3020
139 IFC=39THENA$="DAA":GOTO3010
140 IFC=40THENA$="JR Z,ADDR":GOTO3050
```

```
141 IFC=41THENA$="ADD HL,HL":GOTO3010
142 IFC=42THENA$="LD HL,(ADDR)":GOTO3030
143 IFC=43THENA$="DEC HL":GOTO3010
144 IFC=44THENA$="INC L":GOTO3010
145 IFC=45THENA$="DEC L":GOTO3010
146 IFC=46THENA$="LD L,DATA":GOTO3020
147 IFC=47THENA$="CPL":GOTO3010
148 IFC=48THENA$="JR NC,ADDR":GOTO3050
149 IFC=49THENA$="LD SP,DATA":GOTO3030
150 IFC=50THENA$="LD (ADDR),A":GOTO3030
151 IFC=51THENA$="INC SP":GOTO3010
152 IFC=52THENA$="INC (HL)":GOTO3010
153 IFC=53THENA$="DEC (HL)":GOTO3010
154 IFC=54THENA$="LD (HL),DATA":GOTO3020
155 IFC=55THENA$="SCF":GOTO3010
156 IFC=56THENA$="JR C,ADDR":GOTO3050
157 IFC=57THENA$="ADD HL,SP":GOTO3010
158 IFC=58THENA$="LD A,(ADDR)":GOTO3030
159 IFC=59THENA$="DEC SP":GOTO3010
160 IFC=60THENA$="INC A":GOTO3010
161 IFC=61THENA$="DEC A":GOTO3010
162 IFC=62THENA$="LD A,DATA":GOTO3020
630 IFC=63THENA$="CCF":GOTO3010
640 IFC=64THENA$="LD B,B":GOTO3010
650 IFC=65THENA$="LD B,C":GOTO3010
660 IFC=66THENA$="LD B,D":GOTO3010
670 IFC=67THENA$="LD B,E":GOTO3010
680 IFC=68THENA$="LD B,H":GOTO3010
690 IFC=69THENA$="LD B,L":GOTO3010
700 IFC=70THENA$="LD B,(HL)":GOTO3010
710 IFC=71THENA$="LD B,A":GOTO3010
720 IFC=72THENA$="LD C,B":GOTO3010
730 IFC=73THENA$="LD C,C":GOTO3010
740 IFC=74THENA$="LD C,D":GOTO3010
750 IFC=75THENA$="LD C,E":GOTO3010
760 IFC=76THENA$="LD C,H":GOTO3010
770 IFC=77THENA$="LD C,L":GOTO3010
780 IFC=78THENA$="LD C,(HL)":GOTO3010
790 IFC=79THENA$="LD C,A":GOTO3010
800 IFC=80THENA$="LD D,B":GOTO3010
810 IFC=81THENA$="LD D,C":GOTO3010
820 IFC=82THENA$="LD D,D":GOTO3010
830 IFC=83THENA$="LD D,E":GOTO3010
840 IFC=84THENA$="LD D,H":GOTO3010
850 IFC=85THENA$="LD D,L":GOTO3010
860 IFC=86THENA$="LD D,(HL)":GOTO3010
870 IFC=87THENA$="LD D,A":GOTO3010
880 IFC=88THENA$="LD E,B":GOTO3010
890 IFC=89THENA$="LD E,C":GOTO3010
900 IFC=90THENA$="LD E,D":GOTO3010
910 IFC=91THENA$="LD E,E":GOTO3010
920 IFC=92THENA$="LD E,H":GOTO3010
930 IFC=93THENA$="LD E,L":GOTO3010
940 IFC=94THENA$="LD E,(HL)":GOTO3010
950 IFC=95THENA$="LD E,A":GOTO3010
```

```
960 IFC=96THENA$="LD H,B":GOTO3010
970 IFC=97THENA$="LD H,C":GOTO3010
980 IFC=98THENA$="LD H,D":GOTO3010
990 IFC=99THENA$="LD H,E":GOTO3010
1000 IFC=100THENA$="LD H,H":GOTO3010
1010 IFC=101THENA$="LD H,L":GOTO3010
1020 IFC=102THENA$="LD H,(HL)":GOTO3010
1030 IFC=103THENA$="LD H,A":GOTO3010
1040 IFC=104THENA$="LD L,B":GOTO3010
1050 IFC=105THENA$="LD L,C":GOTO3010
1060 IFC=106THENA$="LD L,D":GOTO3010
1070 IFC=107THENA$="LD L,E":GOTO3010
1080 IFC=108THENA$="LD L,H":GOTO3010
1090 IFC=109THENA$="LD L,L":GOTO3010
1100 IFC=110THENA$="LD L,(HL)":GOTO3010
1110 IFC=111THENA$="LD L,A":GOTO3010
1120 IFC=112THENA$="LD (HL),B":GOTO3010
1130 IFC=113THENA$="LD (HL),C":GOTO3010
1140 IFC=114THENA$="LD (HL),D":GOTO3010
1150 IFC=115THENA$="LD (HL),E":GOTO3010
1160 IFC=116THENA$="LD (HL),H":GOTO3010
1170 IFC=117THENA$="LD (HL),L":GOTO3010
1180 IFC=118THENA$="HALT":GOTO3010
1190 IFC=119THENA$="LD (HL),A":GOTO3010
1200 IFC=120THENA$="LD A,B":GOTO3010
1210 IFC=121THENA$="LD A,C":GOTO3010
1220 IFC=122THENA$="LD A,D":GOTO3010
1230 IFC=123THENA$="LD A,E":GOTO3010
1240 IFC=124THENA$="LD A,H":GOTO3010
1250 IFC=125THENA$="LD A,L":GOTO3010
1260 IFC=126THENA$="LD A,(HL)":GOTO3010
1270 IFC=127THENA$="LD A,A":GOTO3010
1280 IFC=128THENA$="ADD A,B":GOTO3010
1290 IFC=129THENA$="ADD A,C":GOTO3010
1300 IFC=130THENA$="ADD A,D":GOTO3010
1310 IFC=131THENA$="ADD A,E":GOTO3010
1320 IFC=132THENA$="ADD A,H":GOTO3010
1330 IFC=133THENA$="ADD A,L":GOTO3010
1340 IFC=134THENA$="ADD A,(HL)":GOTO3010
1350 IFC=135THENA$="ADD A,A":GOTO3010
1360 IFC=136THENA$="ADC A,B":GOTO3010
1370 IFC=137THENA$="ADC A,C":GOTO3010
1380 IFC=138THENA$="ADC A,D":GOTO3010
1390 IFC=139THENA$="ADC A,E":GOTO3010
1400 IFC=140THENA$="ADC A,H":GOTO3010
1410 IFC=141THENA$="ADC A,L":GOTO3010
1420 IFC=142THENA$="ADC A,(HL)":GOTO3010
1430 IFC=143THENA$="ADC A,A":GOTO3010
1440 IFC=144THENA$="SUB B":GOTO3010
1450 IFC=145THENA$="SUB C":GOTO3010
1460 IFC=146THENA$="SUB D":GOTO3010
1470 IFC=147THENA$="SUB E":GOTO3010
1480 IFC=148THENA$="SUB H":GOTO3010
1490 IFC=149THENA$="SUB L":GOTO3010
1500 IFC=150THENA$="SUB (HL)":GOTO3010
```

```
1510 IFC=151THENA$="SUB A":GOTO3010
1520 IFC=152THENA$="SBC A,B":GOTO3010
1530 IFC=153THENA$="SBC A,C":GOTO3010
1540 IFC=154THENA$="SBC A,D":GOTO3010
1550 IFC=155THENA$="SBC A,E":GOTO3010
1560 IFC=156THENA$="SBC A,H":GOTO3010
1570 IFC=157THENA$="SBC A,L":GOTO3010
1580 IFC=158THENA$="SBC A,(HL)":GOTO3010
1590 IFC=159THENA$="SBC A,A":GOTO3010
1600 IFC=160THENA$="AND B":GOTO3010
1610 IFC=161THENA$="AND C":GOTO3010
1620 IFC=162THENA$="AND D":GOTO3010
1630 IFC=163THENA$="AND E":GOTO3010
1640 IFC=164THENA$="AND H":GOTO3010
1650 IFC=165THENA$="AND L":GOTO3010
1660 IFC=166THENA$="AND (HL)":GOTO3010
1670 IFC=167THENA$="AND A":GOTO3010
1680 IFC=168THENA$="XOR B":GOTO3010
1690 IFC=169THENA$="XOR C":GOTO3010
1700 IFC=170THENA$="XOR D":GOTO3010
1710 IFC=171THENA$="XOR E":GOTO3010
1720 IFC=172THENA$="XOR H":GOTO3010
1730 IFC=172THENA$="XOR L":GOTO3010
1740 IFC=174THENA$="XOR (HL)":GOTO3010
1750 IFC=175THENA$="XOR A":GOTO3010
1760 IFC=176THENA$="OR B":GOTO3010
1770 IFC=177THENA$="OR C":GOTO3010
1780 IFC=178THENA$="OR D":GOTO3010
1790 IFC=179THENA$="OR E":GOTO3010
1800 IFC=180THENA$="OR H":GOTO3010
1810 IFC=181THENA$="OR L":GOTO3010
1820 IFC=182THENA$="OR (HL)":GOTO3010
1830 IFC=183THENA$="OR A":GOTO3010
1840 IFC=184THENA$="CP B":GOTO3010
1850 IFC=185THENA$="CP C":GOTO3010
1860 IFC=186THENA$="CP D":GOTO3010
1870 IFC=187THENA$="CP E":GOTO3010
1880 IFC=188THENA$="CP H":GOTO3010
1890 IFC=189THENA$="CP L":GOTO3010
1900 IFC=190THENA$="CP (HL)":GOTO3010
1910 IFC=191THENA$="CP A":GOTO3010
1920 IFC=192THENA$="RET NZ":GOTO3010
1930 IFC=193THENA$="POP BC":GOTO3010
1940 IFC=194THENA$="JP NZ,ADDR":GOTO3030
1950 IFC=195THENA$="JP ADDR":GOTO3030
1960 IFC=196THENA$="CALL NZ,ADDR":GOTO3030
1970 IFC=197THENA$="PUSH BC":GOTO3010
1980 IFC=198THENA$="ADD A,DATA":GOTO3020
1990 IFC=199THENA$="RST 00H":GOTO3010
2000 IFC=200THENA$="RET Z":GOTO3010
2010 IFC=201THENA$="RET":GOTO3010
2020 IFC=202THENA$="JP Z,ADDR":GOTO3030
2030 IFC=203GOTO3999
2040 IFC=204THENA$="CALL Z,ADDR":GOTO3030
2050 IFC=205THENA$="CALL ADDR":GOTO3030
```

```
2060 IFC=206THENA$="ADC A,DATA":GOTO3020
2070 IFC=207THENA$="RST 08H":GOTO3010
2080 IFC=208THENA$="RET NC":GOTO3010
2090 IFC=209THENA$="POP DE":GOTO3010
2100 IFC=210THENA$="JP NC,ADDR":GOTO3030
2110 IFC=211THENA$="OUT (PORT),A":GOTO3020
2120 IFC=212THENA$="CALL NC, ADDR":GOTO3030
2130 IFC=213THENA$="PUSH DE":GOTO3010
2140 IFC=214THENA$="SUB DATA":GOTO3020
2150 IFC=215THENA$="RST 10H":GOTO3010
2160 IFC=216THENA$="RET C":GOTO3010
2170 IFC=217THENA$="EXX":GOTO3010
2180 IFC=218THENA$="JP C,ADDR":GOTO3030
2190 IFC=219THENA$="IN A,(PORT)":GOTO3020
2200 IFC=220THENA$="CALL C,ADDR":GOTO3030
2210 IFC=221GOTO5000
2220 IFC=222THENA$="SBC A,DATA":GOTO3020
2230 IFC=223THENA$="RST 18H":GOTO3010
2240 IFC=224THENA$="RET PO":GOTO3010
2250 IFC=225THENA$="POP HL":GOTO3010
2260 IFC=226THENA$="JP PO,ADDR":GOTO3030
2270 IFC=227THENA$="EX (SP),HL":GOTO3010
2280 IFC=228THENA$="CALL PO,ADDR":GOTO3030
2290 IFC=229THENA$="PUSH HL":GOTO3010
2300 IFC=230THENA$="AND DATA":GOTO3020
2310 IFC=231THENA$="RST 20H":GOTO3010
2320 IFC=232THENA$="RET PE":GOTO3010
2330 IFC=233THENA$="JP (HL)":GOTO3010
2340 IFC=234THENA$="JP PE,ADDR":GOTO3030
2350 IFC=235THENA$="EX DE,HL":GOTO3010
2360 IFC=236THENA$="CALL PE,ADDR":GOTO3030
2370 IFC=237GOTO6000
2380 IFC=238THENA$="XOR DATA":GOTO3020
2390 IFC=239THENA$="RST 28H":GOTO3010
2400 IFC=240THENA$="RET P":GOTO3010
2410 IFC=241THENA$="POP AF":GOTO3010
2420 IFC=242THENA$="JP P,ADDR":GOTO3030
2430 IFC=243THENA$="DI (DISABLE INT)":GOTO3010
2440 IFC=244THENA$="CALL P,ADDR":GOTO3030
2450 IFC=245THENA$="PUSH AF":GOTO3010
2460 IFC=246THENA$="OR DATA":GOTO3020
2470 IFC=247THENA$="RST 30H":GOTO3010
2480 IFC=248THENA$="RET M":GOTO3010
2490 IFC=249THENA$="LD SP,HL":GOTO3010
2500 IFC=250THENA$="JP M,ADDR":GOTO3030
2510 IFC=251THENA$="EI (ENABLE INT)":GOTO3010
2520 IFC=252THENA$="CALL M,ADDR":GOTO3030
2530 IFC=253GOTO7000
2540 IFC=254THENA$="CP DATA":GOTO3020
2550 IFC=255THENA$="RST 38H":GOTO3010
3010 PRINTA,C,A$:A=A+1:GOTO55
3020 DA=PEEK(A+1)
3025 PRINTA,C,A$,DA:A=A+2:GOTO55
3030 DA=PEEK(A+1)+256*PEEK(A+2)
3035 PRINTA,C,A$,DA:A=A+3:GOTO55
```

```
3040 PRINTA,C,A$:A=A+4:GOTO55
3050 DA=PEEK(A+1):IFDA=>128THENDA=DA-127:DA=128-DA:DA=A+1-DA:GOTO3025
3055 DA=A+2+DA:GOTO3025
3060 PRINTA,C;E,A$:A=A+2:GOTO55
3065 DA=PEEK(F)+256*PEEK(G):PRINTA,C;E,A$,DA:A=A+4:GOTO55
3070 PRINTA,C;E,A$;F;")":A=A+3:GOTO55
3075 PRINTA,C;E,A$;F;"),DATA";G:A=A+4:GOTO55
3080 PRINTA,C;E,A$;F;"),";B$:A=A+3:GOTO55
3085 PRINTA,C;E;G,A$;F;")":A=A+4:GOTO55
3090 DA=F+256*G:PRINTA,C;E,A$,DA:A=A+4:GOTO55
3999 E=PEEK(A+1):IFE>127GOTO4128
4000 IFE=0THENA$="RLC B":GOTO3060
4001 IFE=1THENA$="RLC C":GOTO3060
4002 IFE=2THENA$="RLC D":GOTO3060
4003 IFE=3THENA$="RLC E":GOTO3060
4004 IFE=4THENA$="RLC H":GOTO3060
4005 IFE=5THENA$="RLC L":GOTO3060
4006 IFE=6THENA$="RLC (HL)":GOTO3060
4007 IFE=7THENA$="RLC A":GOTO3060
4008 IFE=8THENA$="RRC B":GOTO3060
4009 IFE=9THENA$="RRC C":GOTO3060
4010 IFE=10THENA$="RRC D":GOTO3060
4011 IFE=11THENA$="RRC E":GOTO3060
4012 IFE=12THENA$="RRC H":GOTO3060
4013 IFE=13THENA$="RRC L":GOTO3060
4014 IFE=14THENA$="RRC (HL)":GOTO3060
4015 IFE=15THENA$="RRC A":GOTO3060
4016 IFE=16THENA$="RL B":GOTO3060
4017 IFE=17THENA$="RL C":GOTO3060
4018 IFE=18THENA$="RL D":GOTO3060
4019 IFE=19THENA$="RL E":GOTO3060
4020 IFE=20THENA$="RL H":GOTO3060
4021 IFE=21THENA$="RL L":GOTO3060
4022 IFE=22THENA$="RL (HL):GOTO3060
4023 IFE=23THENA$="RL A":GOTO3060
4024 IFE=24THENA$="RR B":GOTO3060
4025 IFE=25THENA$="RR C":GOTO3060
4026 IFE=26THENA$="RR D":GOTO3060
4027 IFE=27THENA$="RR E":GOTO3060
4028 IFE=28THENA$="RR H":GOTO3060
4029 IFE=29THENA$="RR L":GOTO3060
4030 IFE=30THENA$="RR (HL)":GOTO3060
4031 IFE=31THENA$="RR A":GOTO3060
4032 IFE=32THENA$="SLA B":GOTO3060
4033 IFE=33THENA$="SLA C":GOTO3060
4034 IFE=34THENA$="SLA D":GOTO3060
4035 IFE=35THENA$="SLA E":GOTO3060
4036 IFE=36THENA$="SLA H":GOTO3060
4037 IFE=37THENA$="SLA L":GOTO3060
4038 IFE=38THENA$="SLA (HL)":GOTO3060
4039 IFE=39THENA$="SLA A":GOTO3060
4040 IFE=40THENA$="SRA B":GOTO3060
4041 IFE=41THENA$="SRA C":GOTO3060
4042 IFE=42THENA$="SRA D":GOTO3060
4043 IFE=43THENA$="SRA E":GOTO3060
4044 IFE=44THENA$="SRA H":GOTO3060
```

```
4045 IFE=45THENA$="SRA L":GOTO3060
4046 IFE=46THENA$="SRA (HL)":GOTO3060
4047 IFE=47THENA$="SRA A":GOTO3060
4056 IFE=56THENA$="SRL B":GOTO3060
4057 IFE=57THENA$="SRL C":GOTO3060
4058 IFE=58THENA$="SRL D":GOTO3060
4059 IFE=59THENA$="SRL E":GOTO3060
4060 IFE=60THENA$="SRL H":GOTO3060
4061 IFE=61THENA$="SRL L":GOTO3060
4062 IFE=62THENA$="SRL (HL)":GOTO3060
4063 IFE=63THENA$="SRL A":GOTO3060
4064 IFE=64THENA$="BIT 0,B":GOTO3060
4065 IFE=65THENA$="BIT 0,C":GOTO3060
4066 IFE=66THENA$="BIT 0,D":GOTO3060
4067 IFE=67THENA$="BIT 0,E":GOTO3060
4068 IFE=68THENA$="BIT 0,H":GOTO3060
4069 IFE=69THENA$="BIT 0,L":GOTO3060
4070 IFE=70THENA$="BIT 0,(HL)":GOTO3060
4071 IFE=71THENA$="BIT 0,A":GOTO3060
4072 IFE=72THENA$="BIT 1,B":GOTO3060
4073 IFE=73THENA$="BIT 1,C":GOTO3060
4074 IFE=74THENA$="BIT 1,D":GOTO3060
4075 IFE=75THENA$="BIT 1,E":GOTO3060
4076 IFE=76THENA$="BIT 1,H":GOTO3060
4077 IFE=77THENA$="BIT 1,L":GOTO3060
4078 IFE=78THENA$="BIT 1,(HL)":GOTO3060
4079 IFE=79THENA$="BIT 1,A":GOTO3060
4080 IFE=80THENA$="BIT 2,B":GOTO3060
4081 IFE=81THENA$="BIT 2,C":GOTO3060
4082 IFE=82THENA$="BIT 2,D":GOTO3060
4083 IFE=83THENA$="BIT 2,E":GOTO3060
4084 IFE=84THENA$="BIT 2,H":GOTO3060
4085 IFE=85THENA$="BIT 2,L":GOTO3060
4086 IFE=86THENA$="BIT 2,(HL)":GOTO3060
4087 IFE=87THENA$="BIT 2,A":GOTO3060
4088 IFE=88THENA$="BIT 3,B":GOTO3060
4089 IFE=89THENA$="BIT 3,C":GOTO3060
4090 IFE=90THENA$="BIT 3,D":GOTO3060
4091 IFE=91THENA$="BIT 3,E":GOTO3060
4092 IFE=92THENA$="BIT 3,H":GOTO3060
4093 IFE=93THENA$="BIT 3,L":GOTO3060
4094 IFE=94THENA$="BIT 3,(HL)":GOTO3060
4095 IFE=95THENA$="BIT 3,A":GOTO3060
4096 IFE=96THENA$="BIT 4,B":GOTO3060
4097 IFE=97THENA$="BIT 4,C":GOTO3060
4098 IFE=98THENA$="BIT 4,D":GOTO3060
4099 IFE=99THENA$="BIT 4,E":GOTO3060
4100 IFE=100THENA$="BIT 4,H":GOTO3060
4101 IFE=101THENA$="BIT 4,L":GOTO3060
4102 IFE=102THENA$="BIT 4,(HL)":GOTO3060
4103 IFE=103THENA$="BIT 4,A":GOTO3060
4104 IFE=104THENA$="BIT 5,B":GOTO3060
4105 IFE=105THENA$="BIT 5,C":GOTO3060
4106 IFE=106THENA$="BIT 5,D":GOTO3060
4107 IFE=107THENA$="BIT 5,E":GOTO3060
```

```
4108 IFE=108THENA$="BIT 5,H":GOTO3060
4109 IFE=109THENA$="BIT 5,L":GOTO3060
4110 IFE=110THENA$="BIT 5,(HL)":GOTO3060
4111 IFE=111THENA$="BIT 5,A":GOTO3060
4112 IFE=112THENA$="BIT 6,B":GOTO3060
4113 IFE=113THENA$="BIT 6,C":GOTO3060
4114 IFE=114THENA$="BIT 6,D":GOTO3060
4115 IFE=115THENA$="BIT 6,E":GOTO3060
4116 IFE=116THENA$="BIT 6,H":GOTO3060
4117 IFE=117THENA$="BIT 6,L":GOTO3060
4118 IFE=118THENA$="BIT 6,(HL)":GOTO3060
4119 IFE=119THENA$="BIT 6,A":GOTO3060
4120 IFE=120THENA$="BIT 7,B":GOTO3060
4121 IFE=121THENA$="BIT 7,C":GOTO3060
4122 IFE=122THENA$="BIT 7,D":GOTO3060
4123 IFE=123THENA$="BIT 7,E":GOTO3060
4124 IFE=124THENA$="BIT 7,H":GOTO3060
4125 IFE=125THENA$="BIT 7,L":GOTO3060
4126 IFE=126THENA$="BIT 7,(HL)":GOTO3060
4127 IFE=127THENA$="BIT 7,A":GOTO3060
4128 IFE=128THENA$="RES 0,B":GOTO3060
4199 IFE=199THENA$="SET 0,A":GOTO3060
4247 IFE=247THENA$="SET 6,A":GOTO3060
5000 E=PEEK(A+1):F=PEEK(A+2):G=PEEK(A+3)
5009 IFE=9THENA$="ADD IX,BC":GOTO3060
5025 IFE=25THENA$="ADD IX,DE":GOTO3060
5033 IFE=33THENA$="LD IX,DATA":GOTO3090
5034 IFE=34THENA$="LD (ADDR),IX":GOTO3090
5035 IFE=35THENA$="INC IX":GOTO3060
5041 IFE=41THENA$="ADD IX,IX":GOTO3060
5042 IFE=42THENA$="LD IX,(ADDR)":GOTO3090
5043 IFE=43THENA$="DEC IX":GOTO3060
5052 IFE=52THENA$="INC (IX+":GOTO3070
5053 IFE=53THENA$="DEC (IX+":GOTO3070
5054 IFE=54THENA$="LD (IX+":GOTO3075
5057 IFE=57THENA$="ADD IX,SP":GOTO3060
5070 IFE=70THENA$="LD B,(IX+":GOTO3070
5078 IFE=78THENA$="LD C,(IX+":GOTO3070
5086 IFE=86THENA$="LD D,(IX+":GOTO3070
5094 IFE=94THENA$="LD E,(IX+":GOTO3070
5102 IFE=102THENA$="LD H,(IX+":GOTO3070
5110 IFE=110THENA$="LD L,(IX+":GOTO3070
5112 IFE=112THENA$="LD (IX+":B$="B":GOTO3080
5113 IFE=113THENA$="LD (IX+":B$="C":GOTO3080
5114 IFE=114THENA$="LD (IX+":B$="D":GOTO3080
5115 IFE=115THENA$="LD (IX+":B$="E":GOTO3080
5116 IFE=116THENA$="LD (IX+":B$="H":GOTO3080
5117 IFE=117THENA$="LD (IX+":B$="L":GOTO3080
5119 IFE=119THENA$="LD (IX+":B$="L":GOTO3080
5126 IFE=126THENA$="LD A,(IX+":GOTO3070
5134 IFE=134THENA$="ADD A,(IX+":GOTO3070
5142 IFE=142THENA$="ADC A,(IX+":GOTO3070
5150 IFE=150THENA$="SUB (IX+":GOTO3070
5158 IFE=158THENA$="SBC A,(IX+":GOTO3070
5166 IFE=166THENA$="AND (IX+":GOTO3070
```

```
5174 IFE=174THENA$="XOR (IX+":GOTO3070
5182 IFE=182THENA$="OR (IX+":GOTO3070
5190 IFE=190THENA$="CP (IX+":GOTO3070
5203 IFE=203GOTO8006
5225 IFE=225THENA$="POP IX":GOTO3060
5227 IFE=227THENA$="EX (SP),iX":GOTO3060
5229 IFE=229THENA$="PUSH IX":GOTO3060
5233 IFE=233THENA$="JP (IX)":GOTO3060
5249 IFE=249THENA$="LD SP,IX":GOTO3060
6000 E=PEEK(A+1):F=A+2:G=A+3
6066 IFE=66THENA$="SBC HL,BC":GOTO3060
6068 IFE=68THENA$="NEG":GOTO3060
6075 IFE=75THENA$="LD BC,(ADDR)":GOTO3065
6082 IFE=82THENA$="SBC HL,DE":GOTO3060
6083 IFE=83THENA$="LD (ADDR),DE":GOTO3065
6091 IFE=91THENA$="LD DE,(ADDR)":GOTO3065
6095 IFE=95THENA$="LD A,R":GOTO3060
6098 IFE=98THENA$="SBC HL,HL":GOTO3065
6176 IFE=176THENA$="LDIR":GOTO3065
6184 IFE=184THENA$="LDDR":GOTO3060
7000 'HERE WE ARE IF FIRST BYTE = 253
7005 E=PEEK(A+1):F=PEEK(A+2):G=PEEK(A+3)
7006 IFE<255THENPRINT"SEE vOL. 2 PAGE 20":GOTO3060
7255 IFE=255THENA$="CONSTANT-TABLE-DATA":GOTO3060
8006 IFG=6THENA$="RLC (IX+":GOTO3085
8014 IFG=14THENA$="RRC (IX+":GOTO3085
8022 IFG=22THENA$="RL (IX+":GOTO3085
8030 IFG=30THENA$="RR (IX+":GOTO3085
8038 IFG=38THENA$="SLA (IX+":GOTO3085
8046 IFG=46THENA$="SRA (IX+":GOTO3085
8062 IFG=62THENA$="SRL (IX+":GOTO3085
8070 IFG=70THENA$="BIT 0,(IX+":GOTO3085
8078 IFG=78THENA$="BIT 1,(IX+":GOTO3085
8086 IFG=86THENA$="BIT 2,(IX+":GOTO3085
8094 IFG=94THENA$="BIT 3,(IX+":GOTO3085
8102 IFG=102THENA$="BIT 4,(IX+":GOTO3085
8110 IFG=110THENA$="BIT 5,(IX+":GOTO3085
8118 IFG=118THENA$="BIT 6,(IX+":GOTO3085
8126 IFG=126THENA$="BIT 7,(IX+":GOTO3085
8134 IFG=134THENA$="RES 0,(IX+":GOTO3085
8142 IFG=142THENA$="RES 1,(IX+":GOTO3085
8150 IFG=150THENA$="RES 2,(IX+":GOTO3085
8158 IFG=158THENA$="RES 3,(IX+":GOTO3085
8166 IFG=166THENA$="RES 4,(IX+":GOTO3085
8174 IFG=174THENA$="RES 5,(IX+":GOTO3085
8182 IFG=182THENA$="RES 6,(IX+":GOTO3085
8190 IFG=190THENA$="RES 7,(IX+":GOTO3085
8198 IFG=198THENA$="SET 0,(IX+":GOTO3085
8206 IFG=206THENA$="SET 1,(IX+":GOTO3085
8214 IFG=214THENA$="SET 2,(IX+":GOTO3085
8222 IFG=222THENA$="SET 3,(IX+":GOTO3085
8230 IFG=230THENA$="SET 4,(IX+":GOTO3085
8238 IFG=238THENA$="SET 5,(IX+":GOTO3085
8246 IFG=246THENA$="SET 6,(IX+":GOTO3085
8254 IFG=254THENA$="SET 7,(IX+":GOTO3085
10000 REM 14714 BYTES MEM - NOT COMPACT, BUT FLEET ! ! !
```

DO LINES 100 - UP OF THIS PROGRAM LOOK FAMILIAR ?

They certainly should if you can remember Chapter 1 of Vol. 2.

What Volume 2?

Gridley, you are with us again. Hope you enjoyed your catnap while the rest of us ploughed through pages 12 - 20 of our new disassembler program. You WILL remember that Chapter 1 of Vol. 2 of the "Disassembled Handbook For TRS-80," listed ALL of the 694 Z-80 instructions in decimal (and most in hex too) numerical order starting with NOP at decimal zero and ending with RST 38H at 255. You will also recall that if the instruction had a first byte of 203, 221, 237, or 255 we branched off to pages 14, 18, 19 or 20 to further decode it, thus giving us a total of 694 instructions.

This disassembler is written in exactly the same way using the IF-THEN statements to matchup the Z-80 opcode and operand with the value derived from PEEKing the desired memory location. Let's analyze the program in detail for you Gridley, though everyone else will easily understand it as it is written in BASIC for clarity PLUS we will be using DECIMAL for everything you can imagine, including addresses, MEM locations, and DATA.

Why decimal? Everybody else uses hex. Why be different?

Ok Gridley, more good well thought out questions. I will try to answer them with questions & answers to illustrate the reasons behind doing so.

What number system does Level II use for line numbers?

ANSWER: decimal

What number system does the TRS-80 use for PEEK and POKE?

ANSWER: decimal

What number system does the Editor/Assembler use for line numbers?

ANSWER: decimal

What number system does the Editor/Assembler use for data or memory locations YOU input to it?

ANSWER: decimal, BUT you may use octal or hex if you fancy it.

Why do so many books and authors use hex for virtually everything imaginable when decimal would be so much simpler?

ANSWER: Most of them have not outgrown their 1975 micros that afford the user the pleasure of keying in a binary byte at a time with switches (ugh). Hex is best in THIS instance.

If decimal is so great, how come our editor/assembler uses hexadeciaml for the object code?

ANSWER: habit. It could just as well be decimal, though hex does have its good points too.

Wouldn't it be a lot easier to understand the object codes generated by the editor/assembler if they were displayed in decimal?

ANSWER: it certainly would.

Does any editor/assembler do all these good things in decimal?

ANSWER: yes. I am writing one for a lark.

Gridley: There sure are a lot of queer, as in strange, birds in this hobby, but I've never seen any larks?

ANSWER: Enough, Gridley. I think we've whipped this poor horse long enough. Let's get on with our DECIMAL DISASSEMBLER discussion and not forget the "foot race" (clocked benchmark test) with the other BASIC disassemblers.

WEIRD DECIMAL DISASSEMBLER DISCUSSION:

LINE 30: For speed's sake all variables except 'D' are defined as integers and the clock interrupt turned off if you are using disk. The variable 'D' (as in DA) will be used to calculate DATA and MEM locations that may exceed the allowable integer range of -32768 to +32767, so we will leave it in single precision floating point format.

LINE 35: Asks the user for the beginning and ending MEM locations he/she wishes to disassemble. IF the location is greater than +32767 we merely subtract -65536 from it and ENTER; i.e., forinstance, if we wished to disassemble MEM locations 55536 to 65535 we would have 55536-65536 = -10000 and then a comma ',' and then 65535-65536 = -1, and then ENTER and off it goes. Remember the breakpoint at 32767 and run it twice if you overlap. If this is too tiresome an exercise for you try adding the following line:

```
37 DEFSNGA,B:IFA>32767THENA=A-65536:B=B-65536
```

LINES 40 & 45: Set up our 4 column headings of:

MEMORY ADDRESS	OBJECT CODE	Z-80 INSTRUCT.	DATA OR ADDRESS

The first column is your decimal address. Second column is the decimal object code LESS data or address. Third column is the Z-80 instruction with DATA or ADDRESS printed out after it if appropriate. Fourth column is DATA or ADDRESS in decimal if appropriate. The next page is a typical printout for a MEM location beginning at decimal 9461 = VARPTR.

MEMORY ADDRESS	OBJECT CODE	Z-80 INSTRUCT.	DATA OR ADDRESS
9461	124	LD A,H	
9462	181	OR L	
9463	202	JP Z,ADDR	7754
9466	205	CALL ADDR	2714
9469	225	POP HL	
9470	201	RET	
9471	254	CP DATA	193
9473	202	JP Z,ADDR	10238
9476	254	CP DATA	197
9478	202	JP Z,ADDR	16797
9481	254	CP DATA	200
9483	202	JP Z,ADDR	10185
9486	254	CP DATA	199
9488	202	JP Z,ADDR	16758
9491	254	CP DATA	198
9493	202	JP Z,ADDR	306
9496	254	CP DATA	201
9498	202	JP Z,ADDR	413
9501	254	CP DATA	196
9503	202	JP Z,ADDR	10799
9506	254	CP DATA	190
9508	202	JP Z,ADDR	16725
9511	214	SUB DATA	215
9513	210	JP NC,ADDR	9550
9516	205	CALL ADDR	9013
9519	207	RST 08H	
9520	41	ADD HL,HL	
9521	201	RET	

In all the above MEM locations, the object code LESS either DATA or ADDRESS was a single byte. Had it been decimal 203, 221, 237, or 253 it would have branched off as appropriate and printed out the second and fourth byte (if used) of the object code right after the first byte. Let's take a look at two MEM locations where this occurs.

MEMORY ADDRESS	OBJECT CODE	Z-80 INSTRUCT.	DATA OR ADDRESS
1170	221 117	LD (IX+ 3),L	
1173	221 116	LD (IX+ 4),H	

Had there been an additional byte in the object code, such as that for 'BIT 1,(IX+zz)' which = 221, 203, zz, 78, then the 78 would have been printed just to the right of the 203 in the object code column. See Volume 2, page 18. For the sake of clarity, zz is printed out INSIDE the parentheses rather than in the last column.

Line 50: Is a weasel way of accomodating MEM location zero, should you wish to START at this location since we have chosen to put our MEM location counter in lines 3010 - 3090. Remove it if you wish.

LINE 55: Ends the program wherever you told it to.

LINE 60: Assigns variable 'A' the value of the MEM location.

LINE 65: Is our 'sneaky Pete' way of speeding up all of those IF-THENs, as mentioned earlier in this Chapter. It quite simply divides them all into 32 line, line segments so that program execution time is enhanced. Try removing this line and benchmarking a RUN from MEM location zero to 1000. It sure slowed down, didn't it. Thirty two line segments is about the opitimum number for maximum program speed. Further subdivision does NOT significantly increase program speed since the additional IF-THENs take time too.

LINES 100 - 2550: Are similar to the first decimal number of our Z-80's object codes. Codes zero through 62 are in lines 100 to 162, and codes 63 through 255 in lines 630 to 2550. This mental gymnastic feat accomodates your author's club-fingered typing prowess. It takes a really determined effort to screw-up typing in the WRONG object code in the WRONG line number and if you do so, it is readily apparent.

LINES 3010 - 3090: Are the real draft horses of this program. Each of the previous lines from 100 to 2550, and subroutines if used, wind-up here for PRINTing and DATA/ADDRESS calculation if necessary. If the object code is only a single byte long, line 3010 handles it. For a 2 byte object code such as CP data, lines 3020 and 3025 fetch the data and print it. Lines 3030 and 3035 do much the same thing for a 3 byte object code where the last 2 bytes contain DATA or an ADDRESS in MEM. You will remember line 3030's general purpose algorithm from Vol. 2. Lines 3050, 3055, and 3060 accomodate our relative jump instruction in rather unsophisticated fashion (as given in Vol. 2), but it works very well, thank you. Lines 3065 to 3090 do much the same thing as 3010 to 3060 except for some of the more weird instructions utilizing the index register IX.

LINES 7006 and 7255: Are a general purpose trap for most of the stranger combinations ginned up by CONSTANTS-TABLES-DATA LISTS and so forth. One or the other lines should catch most of them. Please modify as suits your fancy.

NOTE:
For brevity's sake we have NOT included EVERY POSSIBLE Z-80 instruction in this disassembler. Most of the RES and SET instructions from Vol. 2's pages 15 and 16 have been left out as they are not used by Level II ROM or RAM and NEWDOS+. The same is true for index register IY which is never used. Only those instructions from Vol. 2's page 19 (if first byte = 237) are included that are used by Level II and NEWDOS+. If you wish to add to this disassembler so that ALL possible Z-80 instructions are included, feel free do so. We have NOT done so to ensure that the program would easily fit into TRS-80s with only 16K memory. This program as it stands utilizes 14834 bytes which allows the 16K MEM user about 1K of MEM for an object code program or running room, if desired.

WHEN WE GONNA START THAT BASIC DISASSEMBLER FOOT RACE ? ? ? ?

Right now, Gridley. Here are the general rules of the game.

- THE GREAT BASIC DISASSEMBLER FASTEST FOOT RACE -

RULES:

1. The program MUST be written in BASIC. No sneaky machine language READ-DATA or fake string PEEKs or POKEs are allowed, whatsoever. The author will be the referee on questionable rule breaking. Fastest time wins.
2. The foot race is open to any and all comers. Those that wish to have their programs timed may submit the program on disk to Richcraft Engineering Ltd. with a $10. entry fee to cover prize $ and return postage. All entries must be postmarked prior to September 30, 1980.
3. Run time measurement may be made on any or all parts in MEM from decimal zero to decimal 12288 using the TRS-80 standard clock. Minimum MEM locations disassembled will be 6000. Every disassembler error will ADD 30 seconds to the timed run. At the very minimum, as many Z-80 object codes as utilized by Level II ROM/RAM and NEWDOS+ must be accomodated. Output may be in decimal or hex and MUST include at least: MEM address, 1st byte of object code & complete Z-80 instruction.
4. The winner and losers will be notified by AIRMAIL not later than October 15, 1980.
5. First prize will be greater than $100, second prize greater than $50. and third prize greater than $25. ALL entry fees above actual postage and printing costs will be deposited in the winners' purse and paid out as prize money. Richcraft Engineering Ltd. is guaranteeing minimum prize money.
6. The winner's name will be announced in 80 Microcomputing magazine and if the program is published, the writer will be compensated at the publisher's standard rates.

- -

- ENTRY BLANK -

RICHCRAFT ENGINEERING LTD.
Drawer 1065, #1 Wahmeda Industrial Park
Chautauqua, New York 14722

I wish to shoot crap with destiny. Enclosed is my $10. entry fee and program on mini-disk (35 or 40 track). I understand that you will return it to me within 15 days after receipt.

NAME ______________________________

STREET ______________________________

CITY ____________________ STATE ________ ZIP __________

Richcraft agrees to respect the implied copyright of all contestants' contest entries. None will be published without the author's written permission. All Richcraft employees and their relatives are INELIGIBLE.

WHO'S GONNA SPEND $10 AND ENTER A CRAZY CONTEST LIKE THIS ? ?

Well Gridley, you never know unless you ask. We sent all the major computer magazines and newsletters an individually typed letter requesting they announce the contest and publish the entry blank. We will just have to wait and see. We expect many entries, so first prize money could very well exceed the $1000 level. That kind of prize is certainly not a bad return for the $10 entry fee. If only a few enter the contest then Richcraft puts up the money and everyone wins. If no one enters, then we send the prizes to Reverend Blank and Mr. Abear. Meanwhile, let's get on with our mini-contest.

HERE IS THE COMPETITION:

1. Gerald Abear's Basic Disassembler that is sold by Bill McLaughlin's "S-80 Bulletin" and "Computer Information Exchange" for the munificent sum of $8.00 postpaid. It is listed as CIE People's Software Tape #7 and available from CIE at P. O. Box 158, San Luis Rey, Calif. 92068. This is undoubtedly the MOST cost-effective disassembler on the market. It works, and works very well.

2. Rev. George Blank's Basic Disassembler that is part of Softside's new book, "Pathways Through The ROM." This excellent book is available for $19.95 from Softside Publications, 6 South Street, Milford, N.H. 03055. This is a first rate Basic disassembler, as are ALL the myriad programs written by Pastor Blank. We have deleted one line so it will not pause after 15 lines are output.

Since this is only a mini-contest, we will only time each disassembler's output to video for ROM locations zero to 1000. To be as 'fair' as possible, we will NOT count initializing time, but ONLY that time after MEM location zero is output till finished.

THEY ARE OFF AND RUNNING. ALL BETTING WINDOWS ARE CLOSED.

- -

RACE RESULTS:

1ST PLACE: DISASSEMBLED HANDBOOK DISASSEMBLER
TIME: 1 MINUTES 55 SECONDS

2ND PLACE: REVEREND GEORGE BLANK'S DISASSEMBLER
TIME: 8 MINUTES 55 SECONDS

3RD PLACE: MR. GERALD ABEAR'S DISASSEMBLER
TIME: 137 MINUTES 46 SECONDS

Surely we all know that speed is NOT the whole story. Both the 2nd and 3rd place programs above have redeeming features of very significant value. By the end of this Chapter we will try to cover them, plus look at another approach to the SPEED puzzle. Chapter 2 includes the BIG CONTEST WINNING program.

SIMPLICITY VERSUS SPEED:
Our program on pages 12 through 20 is about as simple in concept, function, and operation as an early Chapter in Doc Lien's excellent "User's Manual For Level 1." Surely the good Professor's advice of "KISS," keep-it-simple-stupid, will usually result in a logical program with relatively fast execution time. Let's see if we can simplify it further, reduce memory requirement, and then you can measure the effect of these changes on execution time.

ANOTHER APPROACH TO SIMPLIFICATION:
Obviously, we could set up a batch of READ - DATA statements which would surely "win" the minimum MEM prize, but our program's execution time would fall about an hour or so BEHIND the proverbial tortoise in the foot race. Let's see if we can have our cake and eat it too? Surely it has occurred to many of you readers that we could calculate the program line number directly from the value of "C" when C = the decimal value of the first byte of the object code (see page 12, line 60). You will recall that our program line numbers are directly related to the decimal value of the first byte of the object code; i.e.:

IF first byte = zero to 62 then line number = 100 to 162
IF first byte = 63 to 255 then line number = 630 to 2550

There are all sorts of ways we can convert "C" to separate digits representing the line number desired. Two of these options are:

1. Using our BASIC string manipulating functions of STR$, LEN, MID$, and VAL to 'peel-off' each digit of "C."

2. Using our BASIC INT function in a short routine to 'peel-off' each digit of "C."

Once we have our line number broken down into separate digits it is a simple matter to tell our program where to go. How so, you say? Well, one way would be to have a dummy GOTO XXXX in an early segment of the program and POKE the desired address into into MEM right after the GOTO instruction.

ISN'T THAT BREAKING THE FASTEST FOOT RACE RULES?

No, Gridley. We specifically OUTLAWED sneaky POKING of machine language READ - DATA or phoney strings into the program, but I think the 4 POKEs we will use here will be allowed. Many more than that would certainly be questionable, in the referee's opinion.

AREN'T YOU THE REFEREE?

I sure am. Now, let's continue.

Will the above scheme work? Sure it will IF we place the dummy GOTO line number in the beginning of the program BEFORE the location of any variables of undetermined size or length.

How do we FIND the GOTO's MEM address and what values are used for the GOTO address? You will remember from page 10 that BASIC's GOTO function = 141 decimal. Try this litte mini-program if you are using disk:

```
10 ' A TRULY WEIRD  B A S I C  DISASSEMBLER BY W4UCH/2 - WEIRD
15 '
20 ' COPYRIGHT (C) 1980 BY RICHCRAFT ENGINEERING LTD.
25 '
27 DEFINTA-C,E-Z:GOTO30
28 GOTO1000
30 FORX=26810TO27000:Y=PEEK(X)
35 IFY=141THENPRINTX;"= GOTO";PEEK(X+1);PEEK(X+2);PEEK(X+3);PEEK(X+4)
40 NEXT
              >RUN
               26962 = GOTO 51  48  0  96
               26970 = GOTO 49  48  48  48
```

If you are using a non-disk system change line 30 to read:

```
30 FORX=17129TO17329:Y=PEEK(X)

              >RUN
               17281 = GOTO 51  48  0  143
               17289 = GOTO 49  48  48  48
```

BEWARE: for the foregoing MEM locations to be CORRECT, lines 10 through 30 must be entered EXACTLY-EXACTLY-EXACTLY the same as shown above. Naturally, any more or any less characters or spaces BEFORE line 30 will change its location in MEM. Note that there are 2 spaces before and after B A S I C.

The first GOTO above is of course the one in line 27. Let's ignore it as the second one is of interest to us. BASIC quite obviously stores the line number in ASCII as 49 48 48 48 = 1000. As such, the MEM locations we wish to use for disk are:

```
49 = 26970 + 1 = 26971 for the first digit
48 = 26970 + 2 = 26972 for the second digit
48 = 26970 + 3 = 26973 for the third digit
48 = 26970 + 4 = 26974 for the fourth digit
```

The MEM locations you should use for non-disk are:

```
49 = 17289 + 1 = 17290 for the first digit
48 = 17289 + 2 = 17291 for the second digit
48 = 17289 + 3 = 17292 for the third digit
48 = 17289 + 4 = 17293 for the fourth digit
```

The following two program excerpts illustrate how we may use either the string manipulation or INT functions to 'peel-off' each digit of our object code's first byte and then POKE its ASCII value into the correct MEM location. Note that if we have a 3 digit line number we POKE decimal 48 = zero into the first MEM location, and if we have a 4 digit line number we POKE decimal 48 = zero into the last MEM location. Change MEM locations as listed above for non-disk or BASIC 2.

USING STRING MANIPULATION TO POKE CORRECT LINE NUMBER:

```
10 ' A TRULY WEIRD  B A S I C  DISASSEMBLER BY W4UCH/2 - WEIRD
15 '
20 ' COPYRIGHT (C) 1980 BY RICHCRAFT ENGINEERING LTD.
25 '
27 DEFINTA-C,E-Z:GOTO30
28 GOTO1000
30 INPUT"MEM BEGINNING , ENDING IN DECIMAL";A,B:CLS:CMD"T"
35 PRINT"MEMORY","OBJECT","Z-80","DATA OR"
40 PRINT"ADDRESS","CODE","INSTRUCT.","ADDRESS"
43 IFA=0THENPRINT" 0"," 243","DI (DISABLE INT)":A=A+1
45 IFA>BGOTO45
50 C=PEEK(A):IFC>99GOTO70
60 IFC<63THENX=C+100:POKE26971,48:MM=26971:GOTO80
65 IFC<100THENX=C*10:POKE26971,48:MM=26971:GOTO80
70 X=C*10:MM=26970
80 X$=STR$(X):FORK=2TOLEN(X$):D$=MID$(X$,K,1):L=VAL(D$)+48
90 MM=MM+1:POKEMM,L:NEXTK:GOTO28
```

USING THE INT FUNCTION TO POKE CORRECT LINE NUMBER:

```
10 ' A TRULY WEIRD  B A S I C  DISASSEMBLER BY W4UCH/2 - WEIRD
15 '
20 ' COPYRIGHT (C) 1980 BY RICHCRAFT ENGINEERING LTD.
25 '
27 DEFINTA-C,E-Z:GOTO30
28 GOTO1000
30 INPUT"MEM BEGINNING , ENDING IN DECIMAL";A,B:CLS:CMD"T"
35 PRINT"MEMORY","OBJECT","Z-80","DATA OR"
40 PRINT"ADDRESS","CODE","INSTRUCT.","ADDRESS"
43 IFA=0THENPRINT" 0"," 243","DI (DISABLE INT)":A=A+1
45 IFA>BGOTO45
50 C=PEEK(A):IFC>99GOTO70
60 IFC<63THENPOKE26971,48:MM=26972:C1=C+100:GOTO75
62 IFC<100THENPOKE26971,48:MM=26972:C1=C*10:GOTO75
65 IFC<100THEN26971,48:MM=26972:C1=C*10:GOTO75
70 C1=C:MM=26971:POKEMM+3,48
75 N=C1:N1=INT(N/100):IFN1>0THENN=N-(N1*100)
80 N2=INT(N/10):IFN2>0THENN=N-(N2*10)
85 N3=N:POKEMM,N1+48:POKEMM+1,N2+48:POKEMM+2,N3+48:GOTO28
```

The above introductory lines to our disassembler program each calculate the correct line number for the first object code byte that line 50 PEEKed and assigned to the variable "C", and then POKEs it into the proper MEM location for line 28's GOTO. Line 28's dummy GOTO value may be any 4 digit number.

WHAT DOES ALL THIS FLIM-FLAMMERY BUY US?
Namely, about 1600 bytes of MEM saved since we do NOT have to include the "IF C=XXXTHEN" in each line from 100 through 2550. Program speed with either approach? You will have to find out for yourself. This is supposed to be a teaching program, not a cookbook full of recipes. Hopefully, we have aroused your curiosity sufficiently so that you will time each program.

COULD WE HAVE SAVED EVEN MORE MEM?
Sure we could. About another 3200 bytes IF we had used the same scheme for calculating the line number of those multi-byte object codes whose first byte is 203, 221, 237 or 253. Feel free to do so, if you have the time and patience.

The following program may be ADDED on to either program given on page 29. Lines 3999 on up are the same as the original program, unless you wish to modify them.

```
100 A$="NOP":GOTO3010
101 A$="LD BC,DATA":GOTO3030
102 A$="LD (BC),A":GOTO3010
103 A$="INC BC":GOTO3010
104 A$="INC B":GOTO3010
105 A$="DEC B":GOTO3010
106 A$="LD B,DATA":GOTO3020
107 A$="RLCA":GOTO3010
108 A$="EX AF,AF'":GOTO3010
109 A$="ADD HL,BC":GOTO3010
110 A$="LD A,(BC)":GOTO3010
111 A$="DEC BC":GOTO301s
112 A$="INC C":GOTO3010
113 A$="DEC C":GOTO3010
114 A$="LD C,DATA":GOTO3020
115 A$="RRCA":GOTO3010
116 A$="DJNZ":GOTO3050
117 A$="LD DE,DATA":GOTO3030
118 A$="LD (DE),A":GOTO3010
119 A$="INC DE":GOTO3010
120 A$="INC D":GOTO3010
121 A$="DEC D":GOTO3010
122 A$="LD D,DATA":GOTO3020
123 A$="RLA":GOTO3010
124 A$="JR ADDR":GOTO3050
125 A$="ADD HL,DE":GOTO3010
126 A$="LD A,(DE)":GOTO3010
127 A$="DEC DE":GOTO3010
128 A$="INC E":GOTO3010
129 A$="DEC E":GOTO3010
130 A$="LD E,DATA":GOTO3020
131 A$="RRA":GOTO3010
132 A$="JR NZ,ADDR":GOTO3050
133 A$="LD HL,DATA":GOTO3030
134 A$="LD (ADDR),HL":GOTO3030
135 A$="INC HL":GOTO3010
136 A$="INC H":GOTO3010
137 A$="DEC H":GOTO3010
138 A$="LD H,DATA":GOTO3020
139 A$="DAA":GOTO3010
140 A$="JR Z,ADDR":GOTO3050
141 A$="ADD HL,HL":GOTO3010
142 A$="LD HL,(ADDR)":GOTO3030
143 A$="DEC HL":GOTO3010
144 A$="INC L":GOTO3010
145 A$="DEC L":GOTO3010
146 A$="LD L,DATA":GOTO3020
147 A$="CPL":GOTO3010
148 A$="JR NC,ADDR":GOTO3050
149 A$="LD SP,DATA":GOTO3030
150 A$="LD (ADDR),A":GOTO3030
151 A$="INC SP":GOTO3010
152 A$="INC (HL)":GOTO3010
153 A$="DEC (HL)":GOTO3010
154 A$="LD (HL),DATA":GOTO3020
155 A$="SCF":GOTO3010
156 A$="JR C,ADDR":GOTO3050
157 A$="ADD HL,SP":GOTO3010
158 A$="LD A,(ADDR)":GOTO3030
159 A$="DEC SP":GOTO3010
160 A$="INC A":GOTO3010
161 A$="DEC A":GOTO3010
162 A$="LD A,DATA":GOTO3020
630 A$="CCF":GOTO3010
640 A$="LD B,B":GOTO3010
650 A$="LD B,C":GOTO3010
660 A$="LD B,D":GOTO3010
670 A$="LD B,E":GOTO3010
680 A$="LD B,H":GOTO3010
690 A$="LD B,L":GOTO3010
700 A$="LD B,(HL)":GOTO3010
710 A$="LD B,A":GOTO3010
720 A$="LD C,B":GOTO3010
730 A$="LD C,C":GOTO3010
740 A$="LD C,D":GOTO3010
750 A$="LD C,E":GOTO3010
760 A$="LD C,H":GOTO3010
770 A$="LD C,L":GOTO3010
780 A$="LD C,(HL)":GOTO3010
790 A$="LD C,A":GOTO3010
800 A$="LD D,B":GOTO3010
810 A$="LD D,C":GOTO3010
820 A$="LD D,D":GOTO3010
830 A$="LD D,E":GOTO3010
840 A$="LD D,H":GOTO3010
850 A$="LD D,L":GOTO3010
860 A$="LD D,(HL)":GOTO3010
870 A$="LD D,A":GOTO3010
880 A$="LD E,B":GOTO3010
890 A$="LD E,C":GOTO3010
900 A$="LD E,D":GOTO3010
910 A$="LD E,E":GOTO3010
```

```
920 A$="LD E,H":GOTO3010
930 A$="LD E,L":GOTO3010
940 A$="LD E,(HL)":GOTO3010
950 A$="LD E,A":GOTO3010
960 A$="LD H,B":GOTO3010
970 A$="LD H,C":GOTO3010
980 A$="LD H,D":GOTO3010
990 A$="LD H,E":GOTO3010
1000 A$="LD H,H":GOTO3010
1010 A$="LD H,L":GOTO3010
1020 A$="LD H,(HL)":GOTO3010
1030 A$="LD H,A":GOTO3010
1040 A$="LD L,B":GOTO3010
1050 A$="LD L,C":GOTO3010
1060 A$="LD L,D":GOTO3010
1070 A$="LD L,E":GOTO3010
1080 A$="LD L,H":GOTO3010
1090 A$="LD L,L":GOTO3010
1100 A$="LD L,(HL)":GOTO3010
1110 A$="LD L,A":GOTO3010
1120 A$="LD (HL),B":GOTO3010
1130 A$="LD (HL),C":GOTO3010
1140 A$="LD (HL),D":GOTO3010
1150 A$="LD (HL),E":GOTO3010
1160 A$="LD (HL),H":GOTO3010
1170 A$="LD (HL),L":GOTO3010
1180 A$="HALT":GOTO3010
1190 A$="LD (HL),A":GOTO3010
1200 A$="LD A,B":GOTO3010
1210 A$="LD A,C":GOTO3010
1220 A$="LD A,D":GOTO3010
1230 A$="LD A,E":GOTO3010
1240 A$="LD A,H":GOTO3010
1250 A$="LD A,L":GOTO3010
1260 A$="LD A,(HL)":GOTO3010
1270 A$="LD A,A":GOTO3010
1280 A$="ADD A,B":GOTO3010
1290 A$="ADD A,C":GOTO3010
1300 A$="ADD A,D":GOTO3010
1310 A$="ADD A,E":GOTO3010
1320 A$="ADD A,H":GOTO3010
1330 A$="ADD A,L":GOTO3010
1340 A$="ADD A,(HL)":GOTO3010
1350 A$="ADD A,A":GOTO3010
1360 A$="ADC A,B":GOTO3010
1370 A$="ADC A,C":GOTO3010
1380 A$="ADC A,D":GOTO3010
1390 A$="ADC A,E":GOTO3010
1400 A$="ADC A,H":GOTO3010
1410 A$="ADC A,L":GOTO3010
1420 A$="ADC A,(HL)":GOTO3010
1430 A$="ADC A,A":GOTO3010
1440 A$="SUB B":GOTO3010
1450 A$="SUB C":GOTO3010
1460 A$="SUB D":GOTO3010
1470 A$="SUB E":GOTO3010
1480 A$="SUB H":GOTO3010
1490 A$="SUB L":GOTO3010
1500 A$="SUB (HL)":GOTO3010
1510 A$="SUB A":GOTO3010
1520 A$="SBC A,B":GOTO3010
1530 A$="SBC A,C":GOTO3010
1540 A$="SBC A,D":GOTO3010
1550 A$="SBC A,E":GOTO3010
1560 A$="SBC A,H":GOTO3010
1570 A$="SBC A,L":GOTO3010
1580 A$="SBC A,(HL)":GOTO3010
1590 A$="SBC A,A":GOTO3010
1600 A$="AND B":GOTO3010
1610 A$="AND C":GOTO3010
1620 A$="AND D":GOTO3s10
1630 A$="AND E":GOTO3010
1640 A$="AND H":GOTO3010
1650 A$="AND L":GOTO3010
1660 A$="AND (HL)":GOTO3010
1670 A$="AND A":GOTO3010
1680 A$="XOR B":GOTO3010
1690 A$="XOR C":GOTO3010
1700 A$="XOR D":GOTO3010
1710 A$="XOR E":GOTO3010
1720 A$="XOR H":GOTO3010
1730 A$="XOR L":GOTO3010
1740 A$="XOR (HL)":GOTO3010
1750 A$="XOR A":GOTO3010
1760 A$="OR B":GOTO3010
1770 A$="OR C":GOTO3010
1780 A$="OR D":GOTO3010
1790 A$="OR E":GOTO3010
1800 A$="OR H":GOTO3010
1810 A$="OR L":GOTO3010
1820 A$="OR (HL)":GOTO3010
1830 A$="OR A":GOTO3010
1840 A$="CP B":GOTO3010
1850 A$="CP C":GOTO3010
1860 A$="CP D":GOTO3010
1870 A$="CP E":GOTO3010
1880 A$="CP H":GOTO3010
1890 A$="CP L":GOTO3010
1900 A$="CP (HL)":GOTO3010
1910 A$="CP A":GOTO3010
1920 A$="RET NZ":GOTO3010
1930 A$="POP BC":GOTO3010
1940 A$="JP NZ,ADDR":GOTO3030
1950 A$="JP ADDR":GOTO3030
1960 A$="CALL NZ,ADDR":GOTO3030
1970 A$="PUSH BC":GOTO3010
1980 A$="ADD A,DATA":GOTO3020
1990 A$="RST 00H":GOTO3010
2000 A$="RET Z":GOTO3010
2010 A$="RET":GOTO3010
2020 A$="JP Z,ADDR":GOTO3030
2030 GOTO3999
```

```
2040 A$="CALL Z,ADDR":GOTO3030
2050 A$="CALL ADDR":GOTO3030
2060 A$="ADC A,DATA":GOTO3020
2070 A$="RST 08H":GOTO3010
2080 A$="RET NC":GOTO3010
2090 A$="POP DE":GOTO3010
2100 A$="JP NC,ADDR":GOTO3030
2110 A$="OUT (PORT),A":GOTO3020
2120 A$="CALL NC, ADDR":GOTO3030
2130 A$="PUSH DE":GOTO3010
2140 A$="SUB DATA":GOTO3020
2150 A$="RST 10H":GOTO3010
2160 A$="RET C":GOTO3010
2170 A$="EXX":GOTO3010
2180 A$="JP C,ADDR":GOTO3030
2190 A$="IN A,(PORT)":GOTO3020
2200 A$="CALL C,ADDR":GOTO3030
2210 GOTO5000
2220 A$="SBC A,DATA":GOTO3020
2230 A$="RST 18H":GOTO3010
2240 A$="RET PO":GOTO3010
2250 A$="POP HL":GOTO3010
2260 A$="JP PO,ADDR":GOTO3030
2270 A$="EX (SP),HL":GOTO3010
2280 A$="CALL PO,ADDR":GOTO3030
2290 A$="PUSH HL":GOTO3010
2300 A$="AND DATA":GOTO3020
2310 A$="RST 20H":GOTO3010
2320 A$="RET PE":GOTO3010
2330 A$="JP (HL)":GOTO3010
2340 A$="JP PE,ADDR":GOTO3030
2350 A$="EX DE,HL":GOTO3010
2360 A$="CALL PE,ADDR":GOTO3030
2370 GOTO6000
2380 A$="XOR DATA":GOTO3020
2390 A$="RST 28H":GOTO3010
2400 A$="RET P":GOTO3010
2410 A$="POP AF":GOTO3010
2420 A$="JP P,ADDR":GOTO3030
2430 A$="DI (DISABLE INT)":GOTO3010
2440 A$="CALL P,ADDR":GOTO3030
2450 A$="PUSH AF":GOTO3010
2460 A$="OR DATA":GOTO3020
2470 A$="RST 30H":GOTO3010
2480 A$="RET M":GOTO3010
2490 A$="LD SP,HL":GOTO3010
2500 A$="JP M,ADDR":GOTO3030
2510 A$="EI (ENABLE INT)":GOTO3010
2520 A$="CALL M,ADDR":GOTO3030
2530 GOTO7000
2540 A$="CP DATA":GOTO3020
2550 A$="RST 38H":GOTO3010
```

HOW ABOUT AN OPTION TO LPRINT?

Why not? You already have one if you used the 'JKL' or '123' assembly language program from Volume 2's Chapter 6. Another way to go would be to change all the PRINT statements in the original program's lines 40 to 50 and 3010 to 3090 to LPRINT. Probably the most painless and convenient way to go is to add lines 36, 37, and 38, plus 9100 to 10300 to the original program and let BASIC do the job for you. We have also tacked on a CLEAR 50 to the end of line 30 and moved CLS to line 36.

```
30 DEFINTA-C,E-Z:CMD"T":CLEAR30
35 INPUT"MEM BEGINNING , ENDING IN DECIMAL";A,B
36 INPUT"DO YOU WISH VIDEO OR LPRINT OUT (V/L)";L$:CLS
37 IFL$="L"GOTO9100
38 IFL$="V"GOTO10000
40 LPRINT"MEMORY","OBJECT","Z-80","DATA OR"
45 LPRINT"ADDRESS","CODE","INSTRUCT.","ADDRESS"
50 IFA=0THENLPRINT" 0"," 243","DI (DIS INT)":A=A+1
55 IFA>BGOTO35

9100 FORX=-30866TO-30240:IFPEEK(X)=178THENPOKEX,175
9200 NEXT
9300 FORX=27080TO27215:IFPEEK(X)=178THENPOKEX,175
9400 NEXT:GOTO40
10000 FORX=-30866TO-30240:IFPEEK(X)=175THENPOKEX,178
10100 NEXT
10200 FORX=27080TO27215:IFPEEK(X)=175THENPOKEX,178
10300 NEXT:GOTO40
```

All we are doing here in lines 9100 through 10300 is scanning through MEM in the vicinity of lines 40 - 50 and 3010 - 3090 and swapping LPRINT = 178 for PRINT = 175 (see page 10), or vice versa, whichever you select in line 36. It is an old trick that has been around for a long time. It makes no nevermind whether lines 40, 45, and 50 are entered as LPRINT or PRINT as the program will change them to whichever you desire.

CAUTION:

Do not casually go around poking 175s for 178s, or vice versa, ANYWHERE in the program as some authors tell you, as it will most certainly foul-up a goodly number of MEM locations. If you do not believe that such is the case, try it indescriminately.....then watch it bomb out.

HOW COME WE DON'T HAVE AN ECHO OPTION FOR SIMULTANEOUS LPRINT AND PRINT?

Ok, Gridley. Do you not think this Chapter long enough as it stands? Here is a homework assignment for you: "write a brief assembly language subroutine that intercepts each line of video. When a line of video is complete, the subroutine then LPRINTS out that line before going on to the next video line."

SORRY I ASKED THE QUESTION.

SUMMARY:

This has been a LONG, but fun Chapter for the author. We hope that you enjoyed it too. We have learned that there are many ways to 'skin the disassembler cat' (forgive the expression Harlequin and R/C). You would do well to study the two excellent BASIC disassemblers written by Gerald Abear and Rev. George Blank as program SPEED is certainly not the only measure of a disassembler's excellence. We have tried to illustrate a few approaches to writing a modestly speedy disassembler using BASIC. Are there others? Sure there are. Many are probably MUCH faster than our approach. That's why we ran the contest from May/June 1980 through September 30, 1980.

The FIRST PLACE winning program in the BIG CONTEST is in Chapter 2 as well as the MOST UNIQUE prize winning program. They are both EXCELLENT and worthy of your study and attention.

If you actually sat down at your TRS-80 keyboard and both entered and ran the programs in this Chapter, plus adding those Z-80 instructions that we left out, the probability is nearly 100 percent that you have become more familiar with the 694+ object codes/instructions available to you as an assembly language programmer. As such, you have surely increased your assembly language programming skills significantly.

Do not KNOCK a teaching program just because it is written in BASIC, since in many instances we can learn considerably more about the inner workings and machinations of our TRS-80 using BASIC as the vehicle rather than an assembly language program.

WANT TO DIG DEEPER:

Try using the NEWDOS+ disassembler to disassemble their disassembler program.

WHAT DID HE SAY?

To disassemble itself, Gridley. This is an excellent program that you will learn a great deal from by tracing its logic and program flow. You surely know by now how most any variety of disassembler operates, so have it and GOOD LUCK.

48

CHAPTER 2

FASTEST FOOTRACE CONTEST WINNING PROGRAMS

This chapter includes two of the many fine programs entered in the 'Great BASIC Disassembler Fastest Footrace Contest.'

The first program is that of the FIRST PRIZE winner, Colonel Charles D. House, USAF (Ret). 'Chick' House is no newcomer to writing first rate programs in either BASIC or assembly language as he is a 'professional' programmer that has written a number of cassette and disk programs for Bill McLaughlin's Computer Information Exchange.

It should come as no surprise that out of the first ten places in the contest, five of the authors were professionals that were either computer science professors/engineers or had previously 'sold' their programs commercially. The competition was fierce, but 'Chick' House's program on pages 8-2 through 8-6 was 'head and heels' the fastest and timed at 2 minutes & 20 seconds to disassemble MEM from zero to 6000. It was 1 minute and 6 seconds FASTER than our old friend, John Blair's (Norfolk, Virginia) program which took second place and a flat two minutes faster than the 3rd place winner, Joel Mick from Cinnaminson, New Jersey.

Chick House's program is certainly worth studying in detail. Every BASIC 'go fast' function (known to us) is used, bar none. Remember, the contest did NOT allow USR CALLS, fake string manipulation, or phoney assembly language tricks. It had to be raw BASIC, pure and simple.

The MOST UNIQUE program prize went to H. Woods Martin of Houston, Texas. This prize was truly a NO CONTEST win for Woods in that the program is one of the most beautifully laid out BASIC programs we have ever seen, plus it was the 5th fastest, placing just behind the 4th place winner, Dr. John W. Blattner who was co-author of INSIDE LEVEL II (published by Bryan Mumford's, Mumford Micro Systems, Summerland, Cal.).

We ALL can learn a great deal from Wood's excellent program as it has a great deal MORE to teach us than just pretty printing at its BEST. It starts on page 8-7.

To all of you who entered the contest, a big THANK YOU. EVERYONE received their entry back. The entry fee was used to keep the zillions of 'squirrels' out there in the hinterlands from swamping us with 'nuts'.

```
1 PRINTTIME$:GOTO33
2 CLS:FORN=1TO12288:N=N-1
4 CLS:FORI=0TO992STEP32
5 ONB(PEEK(N))GOTO7,8,9,10,12
6 ONB(PEEK(N))-5GOTO13,14,15,18,25
7 PRINT@I,N;A$(PEEK(N));:N=N+1:NEXT:NEXT:GOTO17
8 PRINT@I,N;A$(PEEK(N));PEEK(N+1)+256*PEEK(N+2);:N=N+3:NEXT:NEXT
:GOTO17
9 PRINT@I,N;A$(PEEK(N));PEEK(N+1);:N=N+2:NEXT:NEXT:NEXT:GOTO17
10 IFPEEK(N+1)<128THENPRINT@I,N;A$(PEEK(N));N+2+PEEK(N+1);:N=N+2
:NEXT:NEXT:GOTO17
11 PRINT@I,N;A$(PEEK(N));N+PEEK(N+1)-254;:N=N+2:NEXT:NEXT:GOTO17
12 PRINT@I,N;:PRINTUSINGA$(PEEK(N));PEEK(N+1)+256*PEEK(N+2);:N=N
+3:NEXT:NEXT:GOTO17
13 PRINT@I,N;:PRINTUSINGA$(PEEK(N));PEEK(N+1);:N=N+2:NEXT:NEXT:G
OTO17
14 PRINT@I,N;A$(PEEK(N+1)+255);:N=N+2:NEXT:NEXT:GOTO17
15 IFE(PEEK(N+1))>0THENPRINT@I,N;:PRINTUSINGE$(PEEK(N+1));PEEK(N
+2)+256*PEEK(N+3);:N=N+4:NEXT:NEXT:GOTO17
16 PRINT@I,N;E$(PEEK(N+1));:N=N+2:NEXT:NEXT:GOTO17
17 PRINT:PRINTTIME$:STOP
18 ONC(PEEK(N+1))GOTO20,21,22,23,24
19 PRINT@I,N;:PRINTUSINGD$(PEEK(N+3));PEEK(N+3);:N=N+4:NEXT:NEXT
:GOTO17
20 PRINT@I,N;C$(PEEK(N+1));:N=N+2:NEXT:NEXT:GOTO17
21 PRINT@I,N;C$(PEEK(N+1));PEEK(N+2)+256*PEEK(N+3);:N=N+4:NEXT:N
EXT:GOTO17
22 PRINT@I,N;:PRINTUSINGC$(PEEK(N+1));PEEK(N+2)+256*PEEK(N+3);:N
=N+4:NEXT:NEXT:GOTO17
23 PRINT@I,N;:PRINTUSINGC$(PEEK(N+1));PEEK(N+2);:N=N+3:NEXT:NEXT
:GOTO17
24 PRINT@I,N;:PRINTUSINGC$(PEEK(N+1));PEEK(N+2);PEEK(N+3);:N=N+4
:NEXT:NEXT:GOTO17
25 IFPEEK(N+1)=255THENPRINT@I,N;F$(PEEK(N+1));:N=N+2:NEXT:NEXT:G
OTO17
26 ONC(PEEK(N+1))GOTO28,29,30,31,32
27 PRINT@I,N;:PRINTUSINGG$(PEEK(N+3));PEEK(N+3);:N=N+4:NEXT:NEXT
:GOTO17
28 PRINT@I,N;F$(PEEK(N+1));:N=N+2:NEXT:NEXT:GOTO17
29 PRINT@I,N;F$(PEEK(N+1));PEEK(N+2)+256*PEEK(N+3);:N=N+4:NEXT:N
EXT:GOTO17
30 PRINT@I,N;:PRINTUSINGF$(PEEK(N+1));PEEK(N+2)+256*PEEK(N+3);:N
=N+4:NEXT:NEXT:GOTO17
31 PRINT@I,N;:PRINTUSINGF$(PEEK(N+1));PEEK(N+2);:N=N+3:NEXT:NEXT
:GOTO17
32 PRINT@I,N;:PRINTUSINGF$(PEEK(N+1));PEEK(N+2);PEEK(N+3);:N=N+4
:NEXT:NEXT:GOTO17
33 DEFINTA-Z:DIMB(256),A$(511),C(256),C$(256),E$(188),A(72),D$(2
54),F$(256),G$(254),E(188)
34 FORI=0TO62:READB(I):NEXT:FORI=63TO193:B(I)=1:NEXT:FORI=194TO2
55:READB(I):NEXT:FORI=1TO40:READA(I):NEXT:FORI=1TO40:READC(A(I))
:NEXT
35 FORI=0TO511:READA$(I):NEXT:FORI=1TO40:READC$(A(I)):NEXT:FORI=
6TO254STEP8:READD$(I):NEXT:FORI=1TO40:READF$(A(I)):NEXT:FORI=6TO
254STEP8:READG$(I):NEXT
```

```
36 FORI=64TO187:READE$(I):NEXT:FORI=67TO123STEP8:READE(I):NEXT:G
OTO2
37 DATA1,2,1,1,1,1,3,1,1,1,1,1,1,1,3,1,4,2,1,1,1,1,3,1,4,1,1,1,1
,1,3,1,4,2,5,1,1,1,3,1,4,1,5,1,1,1,3,1,4,2,5,1,1,1,3,1,4,1,5,1,1
,1,3
38 DATA2,2,2,1,3,1,1,1,2,7,2,2,3,1,1,1,2,6,2,1,3,1,1,1,2,6,2,9,3
,1,1,1,2,1,2,1,3,1,1,1,2,1,2,8,3,1,1,1,2,1,2,1,3,1,1,1,2,1,2,10,
3,1
39 DATA9,25,33,34,35,41,42,43,52,53,54,57,70,78,86,94,102,110,11
2,113,114,115,116,117,119,126,134,142,150,158,166,174,182,190,22
5,227,229,233,249,255
40 DATA1,1,2,3,1,1,3,1,4,4,5,1,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4
,4,4,4,4,4,1,1,1,1,1,1
41 DATA00 NOP,"01 LD   BC,","02 LD   (BC),A",03 INC  BC,04 INC
B,05 DEC  B,"06 LD   B,",07 RLCA,"08 EX   AF,AF'","09 ADD  HL,BC
","0A LD   A,(BC)",0B DEC  BC,0C INC  C,0D DEC  C,"0E LD   C,",0
F RRCA
42 DATA10 DJNZ,"11 LD   DE,","12 LD   (DE),A",13 INC  DE,14 INC
 D,15 DEC  D,"16 LD   D,",17 RLA,"18 JR   ","19 ADD  HL,DE","1A L
D   A,(DE)",1B DEC  DE,1C INC  E,1D DEC  E,"1E LD   E,",1F RRA
43 DATA"20 JR   NZ,","21 LD   HL,","22 LD   (#####),HL",23 INC
HL,24 INC  H,25 DEC  H,"26 LD   H,",27 DAA,"28 JR   Z,","29 ADD
 HL,HL","2A LD   HL,(#####)",2B DEC  HL,2C INC  L,2D DEC  L,"2E
LD   L,",2F CPL
44 DATA"30 JR   NC,","31 LD   SP,","32 LD   (#####),A",33 INC  S
P,34 INC  (HL),35 DEC  (HL),"36 LD   (HL),",37 SCF,"38 JR   C,",
"39 ADD  HL,SP","3A LD   A,(#####)",3B DEC  SP,3C INC  A,3D DEC
 A,"3E LD   A,",3F CCF
45 DATA"40 LD   B,B","41 LD   B,C","42 LD   B,D","43 LD   B,E","
44 LD   B,H","45 LD   B,L","46 LD   B,(HL)","47 LD   B,A","48 LD
   C,B","49 LD   C,C","4A LD   C,D","4B LD   C,E","4C LD   C,H",
"4D LD   C,L","4E LD   C,(HL)","4F LD   C,A"
46 DATA"50 LD   D,B","51 LD   D,C","52 LD   D,D","53 LD   D,E","
54 LD   D,H","55 LD   D,L","56 LD   D,(HL)","57 LD   D,A","58 LD
   E,B","59 LD   E,C","5A LD   E,D","5B LD   E,E","5C LD   E,H",
"5D LD   E,L","5E LD   E,(HL)","5F LD   E,A"
47 DATA"60 LD   H,B","61 LD   H,C","62 LD   H,D","63 LD   H,E","
64 LD   H,H","65 LD   H,L","66 LD   H,(HL)","67 LD   H,A","68 LD
   L,B","69 LD   L,C","6A LD   L,D","6B LD   L,E","6C LD   L,H",
"6D LD   L,L","6E LD   L,(HL)","6F LD   L,A"
48 DATA"70 LD   (HL),B","71 LD   (HL),C","72 LD   (HL),D","73 LD
   (HL),E","74 LD   (HL),H","75 LD   (HL),L",76 HALT,"77 LD   (H
L),A","78 LD   A,B","79 LD   A,C","7A LD   A,D","7B LD   A,E","7
C LD   A,H","7D LD   A,L","7E LD   A,(HL)","7F LD   A,A"
49 DATA"80 ADD  A,B","81 ADD  A,C","82 ADD  A,D","83 ADD  A,E","
84 ADD  A,H","85 ADD  A,L","86 ADD  A,(HL)","87 ADD  A,A","88 AD
C  A,B","89 ADC  A,C","8A ADC  A,D","8B ADC  A,E","8C ADC  A,H",
"8D ADC  A,L","8E ADC  A,(HL)","8F ADC  A,A"
50 DATA90 SUB  B,91 SUB  C,92 SUB  D,93 SUB  E,94 SUB  H,95 SUB
 L,96 SUB  (HL),97 SUB  A,"98 SBC  A,B","99 SBC  A,C","9A SBC  A
,D","9B SBC  A,E","9C SBC  A,H","9D SBC  A,L","9E SBC  A,(HL)","
9F SBC  A,A"
51 DATAA0 AND  B,A1 AND  C,A2 AND  D,A3 AND  E,A4 AND  H,A5 AND
 L,A6 AND  (HL),A7 AND  A,A8 XOR  B,A9 XOR  C,AA XOR  D,AB XOR
E,AC XOR  H,AD XOR  L,AE XOR  (HL),AF XOR  A
```

```
52 DATAB0 OR   B,B1 OR   C,B2 OR   D,B3 OR   E,B4 OR   H,B5 OR
 L,B6 OR   (HL),B7 OR   A,B8 CP   B,B9 CP   C,BA CP   D,BB CP
E,BC CP   H,BD CP   L,BE CP   (HL),BF CP   A
53 DATAC0 RET  NZ,C1 POP  BC,"C2 JP   NZ,",C3 JP   ,"C4 CALL NZ,"
,C5 PUSH BC,"C6 ADD  A,",C7 RST  00H,C8 RET  Z,C9 RET,"CA JP   Z
,",,"CC CALL Z,",CD CALL,"CE ADC  A,",CF RST  08H
54 DATAD0 RET  NC,D1 POP  DE,"D2 JP   NC,","D3 OUT  (###),A","D4
 CALL NC,",D5 PUSH DE,D6 SUB ,D7 RST  10H,D8 RET  C,D9 EXX,"DA J
P   C,","DB IN   A,(###)","DC CALL C,",,"DE SBC  A,",DF RST  18H
55 DATAE0 RET  PO,E1 POP  HL,"E2 JP   PO,","E3 EX   (SP),HL","E4
 CALL PO,",E5 PUSH HL,E6 AND ,E7 RST  20H,E8 RET  PE,E9 JP   (HL
),"EA JP   PE,","EB EX   DE,HL","EC CALL PE,",,EE XOR ,EF RST  2
8H
56 DATAF0 RET  P,F1 POP  AF,"F2 JP   P,",F3 DI,"F4 CALL P,",F5 P
USH AF,F6 OR  ,F7 RST  30H,F8 RET  M,"F9 LD   SP,HL","FA JP   M,
",FB EI,"FC CALL M,",,FE CP  ,FF RST  38H
57 DATACB RLC  B,CB RLC  C,CB RLC  D,CB RLC  E,CB RLC  H,CB RLC
 L,CB RLC  (HL),CB RLC  A
58 DATACB RRC  B,CB RRC  C,CB RRC  D,CB RRC  E,CB RRC  H,CB RRC
 L,CB RRC  (HL),CB RRC  A,CB RL   B,CB RL   C,CB RL   D,CB RL
E,CB RL   H,CB RL   L,CB RL   (HL),CB RL   A
59 DATACB RR   B,CB RR   C,CB RR   D,CB RR   E,CB RR   H,CB RR
 L,CB RR   (HL),CB RR   A
60 DATACB SLA  B,CB SLA  C,CB SLA  D,CB SLA  E,CB SLA  H,CB SLA
 L,CB SLA  (HL),CB SLA  A
61 DATACB SRA  B,CB SRA  C,CB SRA  D,CB SRA  E,CB SRA  H,CB SRA
 L,CB SRA  (HL),CB SRA  A,,,,,,,,,CB SRL  B,CB SRL  C,CB SRL  D,
CB SRL  E,CB SRL  H,CB SRL  L,CB SRL  (HL),CB SRL  A
62 DATA"CB BIT  0,B","CB BIT  0,C","CB BIT  0,D","CB BIT  0,E","
CB BIT  0,H","CB BIT  0,L","CB BIT  0,(HL)","CB BIT  0,A"
63 DATA"CB BIT  1,B","CB BIT  1,C","CB BIT  1,D","CB BIT  1,E","
CB BIT  1,H","CB BIT  1,L","CB BIT  1,(HL)","CB BIT  1,A"
64 DATA"CB BIT  2,B","CB BIT  2,C","CB BIT  2,D","CB BIT  2,E","
CB BIT  2,H","CB BIT  2,L","CB BIT  2,(HL)","CB BIT  2,A"
65 DATA"CB BIT  3,B","CB BIT  3,C","CB BIT  3,D","CB BIT  3,E","
CB BIT  3,H","CB BIT  3,L","CB BIT  3,(HL)","CB BIT  3,A"
66 DATA"CB BIT  4,B","CB BIT  4,C","CB BIT  4,D","CB BIT  4,E","
CB BIT  4,H","CB BIT  4,L","CB BIT  4,(HL)","CB BIT  4,A"
67 DATA"CB BIT  5,B","CB BIT  5,C","CB BIT  5,D","CB BIT  5,E","
CB BIT  5,H","CB BIT  5,L","CB BIT  5,(HL)","CB BIT  5,A"
68 DATA"CB BIT  6,B","CB BIT  6,C","CB BIT  6,D","CB BIT  6,E","
CB BIT  6,H","CB BIT  6,L","CB BIT  6,(HL)","CB BIT  6,A"
69 DATA"CB BIT  7,B","CB BIT  7,C","CB BIT  7,D","CB BIT  7,E","
CB BIT  7,H","CB BIT  7,L","CB BIT  7,(HL)","CB BIT  7,A"
70 DATA"CB RES  0,B","CB RES  0,C","CB RES  0,D","CB RES  0,E","
CB RES  0,H","CB RES  0,L","CB RES  0,(HL)","CB RES  0,A"
71 DATA"CB RES  1,B","CB RES  1,C","CB RES  1,D","CB RES  1,E","
CB RES  1,H","CB RES  1,L"
72 DATA"CB RES  1,(HL)","CB RES  1,A","CB RES  2,B","CB RES  2,C
","CB RES  2,D","CB RES  2,E","CB RES  2,H","CB RES  2,L","CB RE
S  2,(HL)","CB RES  2,A"
73 DATA"CB RES  3,B","CB RES  3,C","CB RES  3,D","CB RES  3,E","
CB RES  3,H","CB RES  3,L","CB RES  3,(HL)","CB RES  3,A"
74 DATA"CB RES  4,B","CB RES  4,C","CB RES  4,D","CB RES  4,E","
CB RES  4,H","CB RES  4,L"
```

```
75 DATA"CB RES  4,(HL)","CB RES  4,A","CB RES  5,B","CB RES  5,C
","CB RES  5,D","CB RES  5,E","CB RES  5,H","CB RES  5,L","CB RE
S  5,(HL)","CB RES  5,A"
76 DATA"CB RES  6,B","CB RES  6,C","CB RES  6,D","CB RES  6,E","
CB RES  6,H","CB RES  6,L","CB RES  6,(HL)","CB RES  6,A"
77 DATA"CB RES  7,B","CB RES  7,C","CB RES  7,D","CB RES  7,E","
CB RES  7,H","CB RES  7,L"
78 DATA"CB RES  7,(HL)","CB RES  7,A","CB SET  0,B","CB SET  0,C
","CB SET  0,D","CB SET  0,E","CB SET  0,H","CB SET  0,L","CB SE
T  0,(HL)","CB SET  0,A"
79 DATA"CB SET  1,B","CB SET  1,C","CB SET  1,D","CB SET  1,E","
CB SET  1,H","CB SET  1,L","CB SET  1,(HL)","CB SET  1,A"
80 DATA"CB SET  2,B","CB SET  2,C","CB SET  2,D","CB SET  2,E","
CB SET  2,H","CB SET  2,L"
81 DATA"CB SET  2,(HL)","CB SET  2,A","CB SET  3,B","CB SET  3,C
","CB SET  3,D","CB SET  3,E","CB SET  3,H","CB SET  3,L","CB SE
T  3,(HL)","CB SET  3,A"
82 DATA"CB SET  4,B","CB SET  4,C","CB SET  4,D","CB SET  4,E","
CB SET  4,H","CB SET  4,L","CB SET  4,(HL)","CB SET  4,A"
83 DATA"CB SET  5,B","CB SET  5,C","CB SET  5,D","CB SET  5,E","
CB SET  5,H","CB SET  5,L"
84 DATA"CB SET  5,(HL)","CB SET  5,A","CB SET  6,B","CB SET  6,C
","CB SET  6,D","CB SET  6,E","CB SET  6,H","CB SET  6,L","CB SE
T  6,(HL)","CB SET  6,A"
85 DATA"CB SET  7,B","CB SET  7,C","CB SET  7,D","CB SET  7,E","
CB SET  7,H","CB SET  7,L","CB SET  7,(HL)","CB SET  7,A"
86 DATA"DD ADD  IX,BC","DD ADD  IX,DE","DD LD   IX,","DD LD   (#
####),IX","DD INC  IX","DD ADD  IX,IX","DD LD   IX,(#####)","DD
DEC  IX"
87 DATA"DD INC  (IX+##)","DD DEC  (IX+##)","DD LD   (IX+##),###"
,"DD ADD  IX,SP","DD LD   B,(IX+##)","DD LD   C,(IX+##)","DD LD
  D,(IX+##)","DD LD   E,(IX+##)"
88 DATA"DD LD   H,(IX+##)","DD LD   L,(IX+##)","DD LD   (IX+##),
B","DD LD   (IX+##),C","DD LD   (IX+##),D","DD LD   (IX+##),E","
DD LD   (IX+##),H","DD LD   (IX+##),L","DD LD   (IX+##),A"
89 DATA"DD LD   A,(IX+##)","DD ADD  A,(IX+##)","DD ADC  A,(IX+##
)","DD SUB  (IX+##)","DD SBC  A,(IX+##)","DD AND  (IX+##)","DD X
OR  (IX+##)","DD OR   (IX+##)","DD CP   (IX+##)"
90 DATA"DD POP  IX","DD EX   (SP),IX","DD PUSH IX","DD JP   (IX)
","DD LD   SP,IX",DD INDEX SPEC / NOT USED
91 DATA"DD RLC  (IX+##)","DD RRC  (IX+##)","DD RL   (IX+##)","DD
 RR   (IX+##)","DD SLA  (IX+##)","DD SRA  (IX+##)",,"DD SRL  (IX
+##)"
92 DATA"DD BIT  0,(IX+##)","DD BIT  1,(IX+##)","DD BIT  2,(IX+##
)","DD BIT  3,(IX+##)","DD BIT  4,(IX+##)","DD BIT  5,(IX+##)","
DD BIT  6,(IX+##)","DD BIT  7,(IX+##)"
93 DATA"DD RES  0,(IX+##)","DD RES  1,(IX+##)","DD RES  2,(IX+##
)","DD RES  3,(IX+##)","DD RES  4,(IX+##)","DD RES  5,(IX+##)","
DD RES  6,(IX+##)","DD RES  7,(IX+##)"
94 DATA"DD SET  0,(IX+##)","DD SET  1,(IX+##)","DD SET  2,(IX+##
)","DD SET  3,(IX+##)","DD SET  4,(IX+##)","DD SET  5,(IX+##)","
DD SET  6,(IX+##)","DD SET  7,(IX+##)"
95 DATA"FD ADD  IY,BC","FD ADD  IY,FE","DD LD   IY,","FD LD   (#
####),IY","FD INC  IY","FD ADD  IY,IY","FD LD   IY,(#####)","FD
DEC  IY"
```

```
96 DATA"FD INC   (IY+##)","FD DEC   (IY+##)","FD LD    (IY+##),###"
,"FD ADD  IY,SP","FD LD    B,(IY+##)","FD LD    C,(IY+##)","FD LD
  D,(IY+##)","FD LD   E,(IY+##)"
97 DATA"FD LD   H,(IY+##)","FD LD    L,(IY+##)","FD LD    (IY+##),
B","FD LD    (IY+##),C","FD LD    (IY+##),D","DD LD    (IY+##),E","
FD LD    (IY+##),H","FD LD    (IY+##),L","FD LD    (IY+##),A"
98 DATA"FD LD   A,(IY+##)","FD ADD  A,(IY+##)","FD ADC  A,(IY+##
)","FD SUB   (IY+##)","FD SBC  A,(IY+##)","FD AND   (IY+##)","FD X
OR  (IY+##)","FD OR    (IY+##)","FD CP    (IY+##)"
99 DATA"FD POP  IY","FD EX    (SP),IY","FD PUSH IY","FD JP    (IY)
","FD LD   SP,IY",FD INDEX SPEC / NOT USED
100 DATA"FD RLC  (IY+##)","FD RRC  (IY+##)","FD RL    (IY+##)","F
D RR    (IY+##)","FD SLA  (IY+##)","FD SRA  (IY+##)",,"FD SRL   (I
Y+##)"
101 DATA"FD BIT  0,(IY+##)","FD BIT  1,(IY+##)","FD BIT  2,(IY+#
#)","FD BIT  3,(IY+##)","FD BIT  4,(IY+##)","FD BIT  5,(IY+##)",
"FD BIT  6,(IY+##)","FD BIT  7,(IY+##)"
102 DATA"FD RES  0,(IY+##)","FD RES  1,(IY+##)","FD RES  2,(IY+#
#)","FD RES  3,(IY+##)","FD RES  4,(IY+##)","FD RES  5,(IY+##)",
"FD RES  6,(IY+##)","FD RES  7,(IY+##)"
103 DATA"FD SET  0,(IY+##)","FD SET  1,(IY+##)","FD SET  2,(IY+#
#)","FD SET  3,(IY+##)","FD SET  4,(IY+##)","FD SET  5,(IY+##)",
"FD SET  6,(IY+##)","FD SET  7,(IY+##)"
104 DATA"ED IN   B,(C)","ED OUT  (C),B","ED SBC  HL,BC","ED LD
 (#####),BC",ED NEG,ED RETN,ED IM   0,"ED LD   I,A","ED IN   C,(
C)","ED OUT  C,(C)","ED ADC  HL,BC","ED LD   BC,(#####)"
105 DATA,ED RETI,,,"ED IN   D,(C)","ED OUT  (C),D","ED SBC  HL,D
E","ED LD   (#####),DE",,,ED IM   1,"ED LD   A,I","ED IN   E,(C)
"
106 DATA"ED OUT  (C),E","ED ADC  HL,DE","ED LD   DE,(#####)",,,E
D IM   2,"ED LD   A,R","ED IN   H,(C)","ED OUT  (C),H","ED SBC
HL,HL",,,,,ED RRD,"ED IN   L,(C)"
107 DATA"ED OUT  (C),L","ED ADC  HL,HL",,,,,ED RLD,,,"ED SBC  HL
,SP","ED LD   (#####),SP",,,,,"ED IN   A,(C)","ED OUT  (C),A","E
D ADC  HL,SP","ED LD   SP,(#####)",,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
,,,,,,
108 DATAED LDI,ED CPI,ED INI,ED OUTI,,,,,ED LDD,ED CPD,ED IND,ED
 OUTD,,,,,ED LDIR,ED CPIR,ED INIR,ED OTIR,,,,,ED LDDR,ED CPDR,ED
 INDR,ED INDR,1,1,1,1,,,1,1
```

```
1 'FASTONE/BAS  E01   23-AUG-80
WOODS MARTIN   5517 STURBRIDGE  HOUSTON TX 77056  713/621-3786
STAND ALONE FAST DISASSEMBLER

10 CLEAR 10000
:DEFSTR O
:DEFINT I
:DEF FNIA(FA)=FA+(FA>32767)*65536
:DIM OB(255),OC(255),OM(202)
:OH="0123456789ABCDEF"
:OT=STRING$(24," ")
:GOSUB 300

20 IT=1
:IB=0
:IE=22
:CLS
:PRINT "INITIALIZING VARIABLES"
:GOTO 50

30 O=""
:IT=0
:CMD"T"
:CLS
:INPUT"FASTONE   DISASSEMBLER = <ENT>   STOP = S";O
:IF O="S" THEN RUN "BREAK"
ELSE INPUT"CLOCK = C   NO CLOCK = <ENT>";O
:IF O="C" THEN IT=1

40 INPUT"START ADDR (DEC)";FS
:INPUT"  END ADDR (DEC)";FE
:IB=FNIA(FS)
:IE=FNIA(FE)

50 IF IT THEN OS=TIME$
:CMD "R"

60 FOR IP=IB TO IE
:IC=IP
:IN=PEEK(IP)
:IF IN>202
THEN IF IN=203 THEN 110
ELSE IF IN=221 THEN 120
ELSE IF IN=237 THEN 150
ELSE IF IN=253 THEN 160

70 IF INSTR(1,OB(IN)," ")=3 PRINT IC;TAB(10);OB(IN)
: GOTO 100

80 IF INSTR(3,OB(IN),"8405") THEN IP=IP+2
: F=256*PEEK(IP)+PEEK(IP-1)
ELSE IF INSTR(3,OB(IN),"20") OR INSTR(3,OB(IN),"05")
  THEN IP=IP+1
: F=PEEK(IP)
ELSE IF INSTR(3,OB(IN),"2E") THEN IP=IP+1
: F=PEEK(IP)+(PEEK(IP)>127)*256+IC+2
```

```
90 PRINT IC;TAB(10);OB(IN);TAB(35);F

100 NEXT
:OE=TIME$
:GOSUB 260
:GOTO 30


110 IP=IP+1
:PRINT IC;TAB(10);OC(PEEK(IP))
:GOTO 100

120 IP=IP+1
:IF PEEK(IP)<>203 THEN IN=INSTR(1,OM(0),CHR$(PEEK(IP)))+I0
:IF IN=I0 LSET OT="DD**      INVALID"
: GOTO 180
ELSE LSET OT=OM(IN)
:IF IN=15 THEN 140
ELSE 180

130 IP=IP+2
:IN=INSTR(2,OM(1),CHR$(PEEK(IP)))+I1
:IF IN=I1 LSET OT="DDCB**** INVALID"
: GOTO 180
ELSE PRINT IC;TAB(10);OM(IN);TAB(35);PEEK(IP-1)
:GOTO 100

140 IP=IP+2
:PRINT IC;TAB(10);OT;TAB(35);PEEK(IP-1);PEEK(IP)
:GOTO 100

150 IP=IP+1
:IN=INSTR(2,OM(2),CHR$(PEEK(IP)))+I2
:IF IN=I2 LSET OT="ED**      INVALID"
: GOTO 180
ELSE LSET OT=OM(IN)
:GOTO 180

160 IP=IP+1
:IF PEEK(IP)<>203 THEN IN=INSTR(2,OM(3),CHR$(PEEK(IP)))+I3
:IF IN=I3 LSET OT="FD**      INVALID"
: GOTO 180
ELSE LSET OT=OM(IN)
:IF IN=143 THEN 140
ELSE 180

170 IP=IP+2
:IN=INSTR(2,OM(4),CHR$(PEEK(IP)))+I4
:IF IN=I3 LSET OT="FDCB**** INVALID"
: GOTO 180
ELSE PRINT IC;TAB(10);OM(IN);TAB(35);PEEK(IP-1)
:GOTO 100

180 IF MID$(OT,5,1)=" " OR MID$(OT,5,1)="*" THEN
 PRINT IC;TAB(10);OT
:GOTO 100
```

```
190 IF INSTR(5,OT,"8405") THEN IP=IP+2
: F=256*PEEK(IP)+PEEK(IP-1)
ELSE IF INSTR(5,OT,"20") OR INSTR(5,OT,"05") THEN IP=IP+1
: F=PEEK(IP)
ELSE IF INSTR(5,OT,"2E") THEN IP=IP+1
: F=PEEK(IP)+(PEEK(IP)>127)*256+IC+2

200 PRINT IC;TAB(10);OT;TAB(35);F
:GOTO 100

260 IF IT=0 THEN INPUT"COMPLETE <ENT>";I
: RETURN
ELSE OY=OS
:GOSUB 270
:FS=FY
:OY=OE
:GOSUB 270
:GOSUB 280
:RETURN

270 FY=VAL(MID$(OY,10,2))*3600+VAL(MID$(OY,13,2))*60+VAL(MID$(OY,16,2))
:RETURN

280 FY=FY-FS
:IM=INT(FY/60)
:IS=FY-IM*60
:PRINT"ELAPSED TIME ";IE-IB+1;"LOCATIONS   ";IM;"MIN   ";IS;"SEC <ENT>";
:INPUT I
:RETURN

300 PRINT"LOADING OPCODE ARRAYS"

310 FOR I=0 TO 255
:READ OB(I)
:IA=INSTR(1,OB(I),"'")
:IF IA MID$(OB(I),IA,1)=","
312 PRINT OB(I)
:NEXT I

320 FOR I=0 TO 255
:READ OC(I)
:IA=INSTR(1,OC(I),"'")
:IF IA MID$(OC(I),IA,1)=","

322 PRINT OC(I)
:NEXT I

330 FOR I=0 TO 202
:READ OM(I)
:IA=INSTR(1,OM(I),"'")
:IF IA MID$(OM(I),IA,1)=","
```

```
332 PRINT OM(I)
:NEXT I
:PRINT"PAUSE TO CALCULATE OFFSETS"
:GOSUB 350
:RETURN

350 I0=3
:I1=37
:I2=74
:I3=131
:I4=165

360 OM(0)=STRING$(71," ")
:OM(1)=STRING$(32," ")
:OM(2)=STRING$(57," ")
:OM(3)=OM(0)
:OM(4)=OM(1)
:LSET OM(0)=CHR$(I0)
:LSET OM(1)=CHR$(I1)
:LSET OM(2)=CHR$(I2)
:LSET OM(3)=CHR$(I3)
:LSET OM(4)=CHR$(I4)

370 IA=1
:FOR I=5 TO 74
:GOSUB 390
:MID$(OM(0),IA,1)=OD
:NEXT

372 IA=1
:FOR I=76 TO 131
:GOSUB 390
:MID$(OM(2),IA,1)=OD
:NEXT

374 IA=1
:FOR I=133 TO 202
:GOSUB 390
:MID$(OM(3),IA,1)=OD
:NEXT

380 IA=1
:FOR I=39 TO 69
:GOSUB 400
:MID$(OM(1),IA,1)=OD
:NEXT
:IA=1
:FOR I=167 TO 197
:GOSUB 400
:MID$(OM(4),IA,1)=OD
:NEXT
:RETURN
```

```
390 IL=4
:GOTO 410
400 IL=8
410 IB=INSTR(1,OH,MID$(OM(I),IL,1))-1
:IB=IB+16*(INSTR(1,OH,MID$(OM(I),IL-1,1))-1)
:OD=CHR$(IB)
:IA=IA+1
:RETURN

500 DATA 00         NOP,018405   LD BC 'NN,02        LD (BC) 'A,03
     INC BC,04        INC B,05        DEC B,0620      LD B 'N,07
    RLCA
501 DATA 08         EX AF 'AF',09       ADD HL 'BC,0A        LD A '
(BC),0B         DEC BC,0C        INC C,0D        DEC C,0E20      LD C
 'N,0F        RRCA
502 DATA 102E       DJNZ DIS,118405   LD DE 'NN,12        LD (DE) '
A,13        INC DE,14        INC D,15        DEC D,1620      LD D 'N
,17         RLA
503 DATA 182E       JR DIS,19        ADD HL 'DE,1A        LD A '(DE)
,1B         DEC DE,1C        INC E,1D        DEC E,1E20      LD E 'N,
1F         RRA
504 DATA 202E       JR NZ 'DIS,218405   LD HL 'NN,228405   LD (NN)
 'HL,23        INC HL,24        INC H,25        DEC H,2620      LD H
 'N,27        DAA
505 DATA 282E       JR Z 'DIS,29        ADD HL 'HL,2A8405   LD HL '
(NN),2B        DEC HL,2C        INC L,2D        DEC L,2E20      LD L
 'N,2F        CPL
506 DATA 302E       JR NC 'DIS,318405   LD SP 'NN,328405   LD (NN)
 'A,33        INC SP,34        INC (HL),35        DEC (HL),3620
 LD (HL) 'N,37        SCF
507 DATA 382E       JR C 'DIS,39        ADD HL 'SP,3A8405   LD A '(
NN),3B        DEC SP,3C        INC A,3D        DEC A,3E20      LD A
'N,3F        CCF
508 DATA 40         LD B 'B,41       LD B 'C,42       LD B 'D,43
     LD B 'E,44       LD B 'H,45       LD B 'L,46       LD B '(H
L),47        LD B 'A
509 DATA 48         LD C 'B,49       LD C 'C,4A       LD C 'D,4B
     LD C 'E,4C       LD C 'H,4D       LD C 'L,4E       LD C '(H
L),4F        LD C 'A
510 DATA 50         LD D 'B,51       LD D 'C,52       LD D 'D,53
     LD D 'E,54       LD D 'H,55       LD D 'L,56       LD D '(H
L),57        LD D 'A
511 DATA 58         LD E 'B,59       LD E 'C,5A       LD E 'D,5B
     LD E 'E,5C       LD E 'H,5D       LD E 'L,5E       LD E '(H
L),5F        LD E 'A
512 DATA 60         LD H 'B,61       LD H 'C,62       LD H 'D,63
     LD H 'E,64       LD H 'H,65       LD H 'L,66       LD H '(H
L),67        LD H 'A
513 DATA 68         LD L 'B,69       LD L 'C,6A       LD L 'D,6B
     LD L 'E,6C       LD L 'H,6D       LD L 'L,6E       LD L '(H
L),6F        LD L 'A
514 DATA 70         LD (HL) 'B,71       LD (HL) 'C,72       LD (HL
) 'D,73        LD (HL) 'E,74       LD (HL) 'H,75       LD (HL) 'L
,76        HALT,77        LD (HL) 'A
```

```
515 DATA 78        LD A 'B,79        LD A 'C,7A        LD A 'D,7B
     LD A 'E,7C        LD A 'H,7D        LD A 'L,7E        LD A '(H
L),7F       LD A 'A
514 DATA 70        LD (HL) 'B,71        LD (HL) 'C,72        LD (HL
) 'D,73       LD (HL) 'E,74        LD (HL) 'H,75        LD (HL) 'L
,76       HALT,77        LD (HL) 'A
515 DATA 78        LD A 'B,79        LD A 'C,7A        LD A 'D,7B
     LD A 'E,7C        LD A 'H,7D        LD A 'L,7E        LD A '(H
L),7F       LD A 'A
516 DATA 80        ADD A 'B,81        ADD A 'C,82        ADD A 'D,8
3       ADD A 'E,84        ADD A 'H,85        ADD A 'L,86        AD
D A '(HL),87       ADD A 'A
517 DATA 88        ADC A 'B,89        ADC A 'C,8A        ADC A 'D,8
B       ADC A 'E,8C        ADC A 'H,8D        ADC A 'L,8E        AD
C A '(HL),8F       ADC A 'A
518 DATA 90        SUB B,91        SUB C,92        SUB D,93        S
UB E,94       SUB H,95        SUB L,96        SUB (HL),97        SU
B A
519 DATA 98        SBC A 'B,99        SBC A 'C,9A        SBC A 'D,9
B       SBC A 'E,9C        SBC A 'H,9D        SBC A 'L,9E        SB
C A '(HL),9F       SBC A 'A
520 DATA A0        AND B,A1        AND C,A2        AND D,A3        A
ND E,A4       AND H,A5        AND L,A6        AND (HL),A7        AN
D A
521 DATA A8        XOR B,A9        XOR C,AA        XOR D,AB        X
OR E,AC       XOR H,AD        XOR L,AE        XOR (HL),AF        XO
R A
522 DATA B0        OR B,B1        OR C,B2        OR D,B3        OR E
,B4       OR H,B5        OR L,B6        OR (HL),B7        OR A
523 DATA B8        CP B,B9        CP C,BA        CP D,BB        CP E
,BC       CP H,BD        CP L,BE        CP (HL),BF        CP A
524 DATA C0        RET NZ,C1        POP BC,C28405    JP NZ 'NN,C384
05   JP NN,C48405   CALL NZ 'NN,C5        PUSH BC,C620      ADD A
'N,C7       RST 0
525 DATA C8        RET Z,C9        RET,CA8405    JP Z 'NN,CB**
2BYTE INSTR,CC8405   CALL Z 'NN,CD8405   CALL NN,CE20      ADC A
'N,CF       RST 08H
526 DATA D0        RET NC,D1        POP DE,D28405    JP NC 'NN,D320
     OUT (N) 'A,D48405   CALL NC 'NN,D5        PUSH DE,D620        S
UB N,D7       RST 10H
527 DATA D8        RET C,D9        EXX,DA8405    JP C 'NN,DB20
IN A '(N),DC8405   CALL C 'NN,DD**       2BYTE INSTR,DE20      SBC
A 'N,DF       RST 18H
528 DATA E0        RET PO,E1        POP HL,E28405    JP PO 'NN,E3
     EX (SP) 'HL,E48405   CALL PO 'NN,E5        PUSH HL,E620
AND N,E7       RST 20H
529 DATA E8        RET PE,E9        JP (HL),EA8405    JP PE 'NN,EB
     EX DE 'HL,EC8405   CALL PE 'NN,ED**      2BYTE INSTR,EE20
   XOR N,EF       RST 28H
530 DATA F0        RET P,F1        POP AF,F28405    JP P 'NN,F3
   DI,F48405   CALL P 'NN,F5        PUSH AF,F620      OR N,F7
  RST 30H
531 DATA F8        RET M,F9        LD SP 'HL,FA8405    JP M 'NN,FB
     EI,FC8405   CALL M 'NN,FD**      2BYTE INSTR,FE20      CP N,
FF       RST 38H
```

```
532 DATA CB00      RLC B,CB01      RLC C,CB02      RLC D,CB03      R
LC E,CB04      RLC H,CB05      RLC L,CB06      RLC (HL),CB07      RL
C A
533 DATA CB08      RRC B,CB09      RRC C,CB0A      RRC D,CB0B      R
RC E,CB0C      RRC H,CB0D      RRC L,CB0E      RRC (HL),CB0F      RR
C A
534 DATA CB10      RL B,CB11      RL C,CB12      RL D,CB13      RL E
,CB14      RL H,CB15      RL L,CB16      RL (HL),CB17      RL A
535 DATA CB18      RR B,CB19      RR C,CB1A      RR D,CB1B      RR E
,CB1C      RR H,CB1D      RR L,CB1E      RR (HL),CB1F      RR A
536 DATA CB20      SLA B,CB21      SLA C,CB22      SLA D,CB23      S
LA E,CB24      SLA H,CB25      SLA L,CB26      SLA (HL),CB27      SL
A A
537 DATA CB28      SRA B,CB29      SRA C,CB2A      SRA D,CB2B      S
RA E,CB2C      SRA H,CB2D      SRA L,CB2E      SRA (HL),CB2F      SR
A A
538 DATA CB30      INVALID,CB31      INVALID,CB32      INVALID,CB33
     INVALID,CB34      INVALID,CB35      INVALID,CB36      INVALID,
CB37      INVALID
539 DATA CB38      SRL B,CB39      SRL C,CB3A      SRL D,CB3B      S
RL E,CB3C      SRL H,CB3D      SRL L,CB3E      SRL (HL),CB3F      SR
L A
540 DATA CB40      BIT 0 'B,CB41      BIT 0 'C,CB42      BIT 0 'D,C
B43      BIT 0 'E,CB44      BIT 0 'H,CB45      BIT 0 'L,CB46      BI
T 0 '(HL),CB47      BIT 0 'A
541 DATA CB48      BIT 1 'B,CB49      BIT 1 'C,CB4A      BIT 1 'D,C
B4B      BIT 1 'E,CB4C      BIT 1 'H,CB4D      BIT 1 'L,CB4E      BI
T 1 '(HL),CB4F      BIT 1 'A
542 DATA CB50      BIT 2 'B,CB51      BIT 2 'C,CB52      BIT 2 'D,C
B53      BIT 2 'E,CB54      BIT 2 'H,CB55      BIT 2 'L,CB56      BI
T 2 '(HL),CB57      BIT 2 'A
543 DATA CB58      BIT 3 'B,CB59      BIT 3 'C,CB5A      BIT 3 'D,C
B5B      BIT 3 'E,CB5C      BIT 3 'H,CB5D      BIT 3 'L,CB5E      BI
T 3 '(HL),CB5F      BIT 3 'A
544 DATA CB60      BIT 4 'B,CB61      BIT 4 'C,CB62      BIT 4 'D,C
B63      BIT 4 'E,CB64      BIT 4 'H,CB65      BIT 4 'L,CB66      BI
T 4 '(HL),CB67      BIT 4 'A
545 DATA CB68      BIT 5 'B,CB69      BIT 5 'C,CB6A      BIT 5 'D,C
B6B      BIT 5 'E,CB6C      BIT 5 'H,CB6D      BIT 5 'L,CB6E      BI
T 5 '(HL),CB6F      BIT 5 'A
546 DATA CB70      BIT 6 'B,CB71      BIT 6 'C,CB72      BIT 6 'D,C
B73      BIT 6 'E,CB74      BIT 6 'H,CB75      BIT 6 'L,CB76      BI
T 6 '(HL),CB77      BIT 6 'A
547 DATA CB78      BIT 7 'B,CB79      BIT 7 'C,CB7A      BIT 7 'D,C
B7B      BIT 7 'E,CB7C      BIT 7 'H,CB7D      BIT 7 'L,CB7E      BI
T 7 '(HL),CB7F      BIT 7 'A
548 DATA CB80      RES 0 'B,CB81      RES 0 'C,CB82      RES 0 'D,C
B83      RES 0 'E,CB84      RES 0 'H,CB85      RES 0 'L,CB86      RE
S 0 '(HL),CB87      RES 0 'A
549 DATA CB88      RES 1 'B,CB89      RES 1 'C,CB8A      RES 1 'D,C
B8B      RES 1 'E,CB8C      RES 1 'H,CB8D      RES 1 'L,CB8E      RE
S 1 '(HL),CB8F      RES 1 'A
```

```
550 DATA CB90      RES 2 'B,CB91     RES 2 'C,CB92      RES 2 'D,C
B93      RES 2 'E,CB94      RES 2 'H,CB95      RES 2 'L,CB96      RE
S 2 '(HL),CB97      RES 2 'A
551 DATA CB98      RES 3 'B,CB99     RES 3 'C,CB9A      RES 3 'D,C
B9B      RES 3 'E,CB9C      RES 3 'H,CB9D      RES 3 'L,CB9E      RE
S 3 '(HL),CB9F      RES 3 'A
552 DATA CBA0      RES 4 'B,CBA1     RES 4 'C,CBA2      RES 4 'D,C
BA3      RES 4 'E,CBA4      RES 4 'H,CBA5      RES 4 'L,CBA6      RE
S 4 '(HL),CBA7      RES 4 'A
553 DATA CBA8      RES 5 'B,CBA9     RES 5 'C,CBAA      RES 5 'D,C
BAB      RES 5 'E,CBAC      RES 5 'H,CBAD      RES 5 'L,CBAE      RE
S 5 '(HL),CBAF      RES 5 'A
554 DATA CBB0      RES 6 'B,CBB1     RES 6 'C,CBB2      RES 6 'D,C
BB3      RES 6 'E,CBB4      RES 6 'H,CBB5      RES 6 'L,CBB6      RE
S 6 '(HL),CBB7      RES 6 'A
555 DATA CBB8      RES 7 'B,CBB9     RES 7 'C,CBBA      RES 7 'D,C
BBB      RES 7 'E,CBBC      RES 7 'H,CBBD      RES 7 'L,CBBE      RE
S 7 '(HL),CBBF      RES 7 'A
556 DATA CBC0      SET 0 'B,CBC1     SET 0 'C,CBC2      SET 0 'D,C
BC3      SET 0 'E,CBC4      SET 0 'H,CBC5      SET 0 'L,CBC6      SE
T 0 '(HL),CBC7      SET 0 'A
557 DATA CBC8      SET 1 'B,CBC9     SET 1 'C,CBCA      SET 1 'D,C
BCB      SET 1 'E,CBCC      SET 1 'H,CBCD      SET 1 'L,CBCE      SE
T 1 '(HL),CBCF      SET 1 'A
558 DATA CBD0      SET 2 'B,CBD1     SET 2 'C,CBD2      SET 2 'D,C
BD3      SET 2 'E,CBD4      SET 2 'H,CBD5      SET 2 'L,CBD6      SE
T 2 '(HL),CBD7      SET 2 'A
559 DATA CBD8      SET 3 'B,CBD9     SET 3 'C,CBDA      SET 3 'D,C
BDB      SET 3 'E,CBDC      SET 3 'H,CBDD      SET 3 'L,CBDE      SE
T 3 '(HL),CBDF      SET 3 'A
560 DATA CBE0      SET 4 'B,CBE1     SET 4 'C,CBE2      SET 4 'D,C
BE3      SET 4 'E,CBE4      SET 4 'H,CBE5      SET 4 'L,CBE6      SE
T 4 '(HL),CBE7      SET 4 'A
561 DATA CBE8      SET 5 'B,CBE9     SET 5 'C,CBEA      SET 5 'D,C
BEB      SET 5 'E,CBEC      SET 5 'H,CBED      SET 5 'L,CBEE      SE
T 5 '(HL),CBEF      SET 5 'A
562 DATA CBF0      SET 6 'B,CBF1     SET 6 'C,CBF2      SET 6 'D,C
BF3      SET 6 'E,CBF4      SET 6 'HH,CBF5      SET 6 'L,CBF6      S
ET 6 '(HL),CBF7      SET 6 'A
563 DATA CBF8      SET 7 'B,CBF9     SET 7 'C,CBFA      SET 7 'D,C
BFB      SET 7 'E,CBFC      SET 7 'H,CBFD      SET 7 'L,CBFE      SE
T 7 '(HL),CBFF      SET 7 'A
564 DATA ,,,,,DD09      ADD IX 'BC,DD19      ADD IX 'DE,DD218405 L
D IX 'NN
565 DATA DD228405 LD (NN) 'IX,DD23      INC IX,DD29      ADD IX 'I
X,DD2A8405 LD IX '(NN),DD2B      DEC IX,DD3405    INC (IX+IND),DD3
505    DEC (IX+IND),DD360520 LD (IX+IND) 'N
566 DATA DD39      ADD IX 'SP,DD4605    LD B '(IX+IND),DD4E05    LD
 C '(IX+IND),DD5605    LD D '(IX+IND),DD5E05    LD E '(IX+IND),DD6
605    LD H '(IX+IND),DD6E05    LD L '(IX+IND),DD7005    LD (IX+IND
) 'B
```

```
567 DATA DD7105   LD (IX+IND) 'C,DD7205   LD (IX+IND) 'D,DD7305
  LD (IX+IND) 'E,DD7405   LD (IX+IND) 'H,DD7505   LD (IX+IND) 'L
,DD7705   LD (IX+IND) 'A,DD7E05   LD A '(IX+IND),DD8605   ADD A
'(IX+IND)
568 DATA DD8E05   ADC A '(IX+IND),DD9605   SUB (IX+IND),DD9E05
 SBC A '(IX+IND),DDA605   AND (IX+IND),DDAE05   XOR (IX+IND),DDB
605   OR (IX+IND),DDBE05   CP (IX+IND),DDCB0506 RLC (IX+IND)
569 DATA DDCB050E RRC (IX+IND),DDCB0516 RL (IX+IND),DDCB051E RR
(IX+IND),DDCB0526 SLA (IX+IND),DDCB052E SRA (IX+IND),DDCB053E SR
L (IX+IND),DDCB0546 BIT 0 '(IX+IND),DDCB054E BIT 1 '(IX+IND)
570 DATA DDCB0556 BIT 2 '(IX+IND),DDCB055E BIT 3 '(IX+IND),DDCB0
566 BIT 4 '(IX+IND),DDCB056E BIT 5 '(IX+IND),DDCB0576 BIT 6 '(IX
+IND),DDCB057E BIT 7 '(IX+IND),DDCB0586 RES 0 '(IX+IND),DDCB058E
 RES 1 '(IX+IND)
571 DATA DDCB0596 RES 2 '(IX+IND),DDCB059E RES 3 '(IX+IND),DDCB0
5A6 RES 4 '(IX+IND),DDCB05AE RES 5 '(IX+IND),DDCB05B6 RES 6 '(IX
+IND),DDCB05BE RES 7 '(IX+IND),DDCB05C6 SET 0 '(IX+IND),DDCB05CE
 SET 1 '(IX+IND)
572 DATA DDCB05D6 SET 2 '(IX+IND),DDCB05DE SET 3 '(IX+IND),DDCB0
5E6 SET 4 '(IX+IND),DDCB05EE SET 5 '(IX+IND),DDCB05F6 SET 6 '(IX
+IND),DDCB05FE SET 7 '(IX+IND),DDE1     POP IX,DDE3     EX (SP)
'HL
573 DATA DDE5     PUSH IX,DDE9     JP (IX),DDF9     LD SP 'IX,,E
D40     IN B '(C),ED41     OUT (C) 'B,ED42     SBC HL 'BC,ED4384
05 LD (NN) 'BC
574 DATA ED44     NEG,ED45     RETN,ED46     IM 0,ED47     LD I
'A,ED48     IN C '(C),ED49     OUT (C) 'C,ED4A     ADC HL 'BC,ED
4B8405 LD BC  '(NN)
575 DATA ED4D     RETI,ED4F     LD R 'A,ED50     IN D '(C),ED51
    OUT (C) 'D,ED52     SBC HL 'DE,ED538405 LD (NN) 'DE,ED56
 IM 1,ED57     LD A 'I
576 DATA ED58     IN E '(C),ED59     OUT (C) 'E,ED5A     ADC HL
'DE,ED5B8405 LD DE '(NN),ED5E     IM 2,ED5F     LD A 'R,ED60
 IN H '(C),ED61     OUT (C) 'H
577 DATA ED62     SBC HL 'HL,ED67     RRD,ED68     L '(C),ED69
   OUT (C) 'L,ED6A     ADC HL 'HL,ED6F     RLD,ED72     SBC HL '
SP,ED738405 LD (NN) 'SP
578 DATA ED78     IN A '(C),ED79     OUT (C) 'A,ED7A     ADC HL
'SP,ED7B8405 LD SP '(NN),EDA0     LDI,EDA1     CPI,EDA2     INI,
EDA3     OUTI
579 DATA EDA8     LDD,EDA9     CPD,EDAA     IND,EDAB     OUTD,ED
B0     LDIR,EDB1     CPIR,EDB2     INIR,EDB3     OTIR
580 DATA EDB8     LDDR,EDB9     CPDR,EDBA     INDR,EDBB     OTDR
,,FD09     ADD IY 'BC,FD19     ADD IY 'DE,FD218405 LD IY 'NN
581 DATA FD228405 LD (NN) 'IY,FD23     INC IY,FD29     ADD IY 'I
Y,FD2A8405 LD IY '(NN),FD2B     DEC IY,FD3405     INC (IY+IND),FD3
505   DEC (IY+IND),FD360520 LD (IY+IND) 'N
582 DATA FD39     ADD IY 'SP,FD4605   LD B '(IY+IND),FD4E05   LD
 C '(IY+IND),FD5605   LD D '(IY+IND),FD5E05   LD E '(IY+IND),FD6
605   LD H '(IY+IND),FD6E05   LD L '(IY+IND),FD7005   LD (IY+IND
) 'B
```

```
583 DATA FD7105   LD (IY+IND) 'C,FD7205   LD (IY+IND) 'D,FD7305
  LD (IY+IND) 'E,FD7405   LD (IY+IND) 'H,FD7505   LD (IY+IND) 'L
,FD7705   LD (IY+IND) 'A,FD7E05   LD A '(IY+IND),FD8605   ADD A
'(IY+IND)
584 DATA FD8E05   ADC A '(IY+IND),FD9605   SUB (IY+IND),FD9E05
 SBC A '(IY+IND),FDA605   AND (IY+IND),FDAE05   XOR (IY+IND),FDB
605   OR (IY+IND),FDBE05   CP (IY+IND),FDCB0506 RLC (IY+IND)
585 DATA FDCB050E RRC (IY+IND),FDCB0516 RL (IY+IND),FDCB051E RR
(IY+IND),FDCB0526 SLA (IY+IND),FDCB052E SRA (IY+IND),FDCB053E SR
L (IY+IND),FDCB0546 BIT 0 '(IY+IND),FDCB054E BIT 1 '(IY+IND)
586 DATA FDCB0556 BIT 2 '(IY+IND),FDCB055E BIT 3 '(IY+IND),FDCB0
566 BIT 4 '(IY+IND),FDCB056E BIT 5 '(IY+IND),FDCB0576 BIT 6 '(IY
+IND),FDCB057E BIT 7 '(IY+IND),FDCB0586 RES 0 '(IY+IND),FDCB058E
 RES 1 '(IY+IND)
587 DATA FDCB0596 RES 2 '(IY+IND),FDCB059E RES 3 '(IY+IND),FDCB0
5A6 RES 4 '(IY+IND),FDCB05AE RES 5 '(IY+IND),FDCB05B6 RES 6 '(IY
+IND),FDCB05BE RES 7 '(IY+IND),FDCB05C6 SET 0 '(IY+IND),FDCB05CE
 SET 1 '(IY+IND)
588 DATA FDCB05D6 SET 2 '(IY+IND),FDCB05DE SET 3 '(IY+IND),FDCB0
5E6 SET 4 '(IY+IND),FDCB05EE SET 5 '(IY+IND),FDCB05F6 SET 6 '(IY
+IND),FDCB05FE SET 7 '(IY+IND),FDE1     POP IY,FDE3     EX (SP)
'IY
589 DATA FDE5     PUSH IY,FDE9     JP (IY),FDF9     LD SP 'IY
```

- End of Woods Martin's MOST UNIQUE prize winning program -

- CHAPTER 3 -

LPRINT FROM FIFO WHILE KEYBOARD INPUTS - SPOOLING

INTRODUCTION:
My what a stange title. After a 'weird' Chapter 1, I guess we should expect most anything! What does it all mean?

Hang loose, Gridley. I will try to explain SPOOLING first. It is a carryover from days 'BD,' that is Before Disk when most everything was stored on magnetic tape. In olden times, the magnetic tapes ran quite slowly. Let us assume baud rates of 300 to 1500 maximum. Even using 'slow' processors the computer literally loafed along while loading or saving data at these slow rates of data transfer. Some engineers brighter than the average bear figured out that the computer could process an enormous amount of data in the idle time BETWEEN loading or saving each bit on the magnetic tape which was mounted on large SPOOLS. Hence the name SPOOLING came to mean 'having the computer doing something useful when it otherwise would be sitting there doing nothing.' Today, it is a generic term that has little to do with its origins. We'll use it in the broad sense and try to demonstrate SPOOLING by having our TRS-80 accept keyboard input at what appears to be the same time it is LPRINTing out data from our simulated FIFO.

How about FIFO? Maybe you have used them or maybe not. If not, here is what FIFOs do. They are neat little chips, usually capable of holding 4 or 8 bits wide by 64 or more bits long (deep). The newer ones are rather fast little rascals with speeds up to 15 MHz (megahertz) not uncommon today. Our FIFO's job is to accept a byte of 7 or 8 bits of data at its input at most any speed the TRS-80 can handle and let it ripple down to the output end where it is 'held' until a clock pulse releases it for output. By adjusting the speed of the clock pulse to the FIFO output, we can easily slowdown, speed-up, or regulate its output at a constant rate. If the output rate is slower than the input, the FIFO will raise a FULL flag when all 64 PARALLEL byte positions are filled that says, 'slowdown a bit, I'm full.'

One of the primary applications for FIFOs is in field of data communications where the data rates MUST meet an accepted standard. Some of the more common standards are those used by computer bulletin boards (CBBS) which allow TRS-80s to swap data over ordinary telephone lines, as well as amateur radio teletype using Baudot or ASCII codes. Assuming that Ma Bell's phone lines will handle data up to 2500 cycles bandwidth, it would be useless to try and send TRS-80 data at a rate that required 100,000 cycles bandwidth over a telephone line. A FIFO (real one or software created FIFO) would very neatly do this slowdown job for us with its output into a UART which is a Universal-Asynchronous-Receiver-Transmitter, that would change our parallel data output into serial output format with start and stop bits for telephone line transmission. Just like a FIFO, a UART may be a hardware CHIP, or software program.

LPRINT WHILE KEYBOARD INPUT PROGRAM IN BASIC ? ?

Why not? Why not, indeed. Let's start off with a very simple demo program in BASIC and then leap upward and onward with a somewhat more sophisticated demo program in assembly language later in this Chapter. This little 4 liner BASIC program illustrates simulated FIFO operation and SPOOLING (sort of), all in one fell swoop by appearing to simultaneously LPRINT data while you input additional data from the keyboard to video memory WITHOUT using the Z-80's interrupt. Both operations are virtually independent of the other since the program uses the time required by the line printer to mechanically print a character to input a few characters from the keyboard. Load the following program.

```
10 'LPRINT WHILE SIMULTANEOUS KEYBOARD INPUT - BAS1
20 '
30 CLS:DEFINTA-Z:LPRINT:B=15360:C=0
40 A$=INKEY$:PRINTA$;:IFPEEK(14400)=64GOTO50ELSE40
50 LPRINTCHR$(PEEK(B));:B=B+1:C=C+1:IFC=64THENC=0:LPRINT
60 A$=INKEY$:PRINTA$;:IFPEEK(14312)=63GOTO50ELSE60
```

Now, type in a few lines from H.M.S. Pinafore (our apologies to Gilbert & Sullivan) and then press the right arrow key on the keyboard to start your line printer clanking away. Keep typing as it LPRINTS.

PROGRAM OUTPUT:
WHEN I WAS A LAD I SERVED A TERM AS AN OFFICE BOY TO AN ATTORN
EY'S FIRM. I WASHED THE WINDOWS AND I SCRUBBED THE FLOORS AND
I POLISHED UP THE HANDLE OF THE BIG FRONT DOOR. (hit the right
arrow about here to start LPRINT and keep typing) NOW LANDSMEN
ALL WHOMEVER YOU MAY BE IF YOU WISH TO CLIMB TO THE TOP OF THE
TREE, AND YOUR SOUL IS NOT FETTERED TO AN OFFICE STOOL JUST LE
ARN TO BE GUIDED BY THIS GOLDEN RULE. STICK CLOSE TO YOUR DESK
S AND NEVER GO TO SEA & YOU ALL MAY RULERS OF THE QUEENS NAVY.
- (this is about where our LPRINTer caught up and passed us) -

Depending on your typing speed and the speed of your line printer, the line printer may or may not catch up with you. This demo program is good for only 16 lines of video, so do not become over enthusiastic.

THe MINI-PROGRAM:

Line 30:
CLS and then a carriage return. DEFINT certainly speeds up our program compared with single precision (as you should remember from Vol. 2). Variable B is set at 15360 = beginning of video MEM and variable C, our line printer's characters per line counter, at zero.

Line 40:
Allows us to enter a few lines of Gilbert & Sullivan before we hit right-arrow to start LPRINTing. MEM location 14400 decimal = our keyboard's MEM location for right arrow = 64 if pressed.

Line 50:
Begins LPRINTing from video MEM. Variable B is incremented one video MEM location after each LPRINT and our characters per line counter C, also incremented. IF C = 64 (end of the line) then C is reset to zero and a LPRINT (carriage return) is output.

Line 60:
Let's us type away at the keyboard UNTIL our line printer's handshake of ASCII 63 = ? (what next? I'm ready for another character) is sent to the linter printer's MEM address at 37E8H = 14312 decimal. IF ready, it GOTOs line 50 to LPRINT another character ELSE back to line 60 for another look at the keyboard.

WHERE'S THE FIFO ? ?

Right in front of you, Gridley. Video MEM is actually 'a sort of' FIFO in this program. Instead of having our 7 or 8 bit byte "ripple-down" to the output end of our FIFO, we're advancing our LPRINT counter +1 via variable B after each video MEM location is LPRINTed.....practically the same thing. Are we clocking data out at a constant rate like a normal FIFO? No, but we could in an assembly language program where execution time is 300+ times faster than BASIC. This mini-program only scans the keyboard UNTIL the line printer's hand-shake of 63 decimal is received. Does our video MEM simulated FIFO have a 'full' flag? No, but we can simulate one by testing variable B and if it = 16384 (end of video MEM) then issue a CLS and reset B at 15360 = beginning of video MEM.

BASIC's execution speed with our 1.77 MHz clock is the limiting factor regarding the number of keyboard scans possible between each LPRINT. If you are a slow typist (like the author), try changing line 60 & add 70 to slow it down:

```
60 A$=INKEY$:PRINTA$;:D=D+1:IFD=10THEND=0:GOTO50
70 GOTO60
```

Want to speed it up? TRY Mumford Micro Systems 3 speed clock mod at $25.95 ppd. It will speed-up the clock a BIG 50%.

SIMULATING FIFO EMPTY:

We could simulate FIFO 'empty' by having our program look ahead at the next 3 MEM locations and if ALL 3 were empty = ASCII 32 = space, then STOP the line printer and wait for further keyboard input (no blank lines please). Try modifying the program as follows:

```
60 A$=INKEY$:PRINTA$;:IFPEEK(B+1)=32ANDPEEK(B+2)=32ANDPEEK(B+3
)=32GOTO60ELSE50
```

When the program catches up with you slow typists (like the author), it will diddle along at whatever speed that suits your fancy.....illustrating what a poor typist you (we) are.

A SIMILAR SPOOLER PROGRAM WRITTEN IN ASSEMBLY LANGUAGE:

Let's have a go at writing a similar LPRINT while keyboard inputs program, but add a few bells and whistles to it since it will execute 300+ times faster than our program written in BASIC. As such, our line printer won't even know we are in the loop inputing data during its mechanically busy printing time. We should remember that this type of subroutine will work with most any variety of program/device that is SLOWER than our subroutine's execution speed, including 300 baud computer bulletin board systems, Morse code, and radio teletype programs where we might wish to have the TYPE AHEAD capabilty WHILE simultaneously receiving data on our split screen video display (see Vol. 2) or transmitting a standard message or program from MEM. In these cases, we could use the time stolen from either a stop bit or timing space to output a character to our line printer and/or video display.

Yet another option is to buy or build a separate FIFO dedicated exclusively to our LPRINTer and possibly load it in parallel with our video RAM memory, or even "burst load" this FIFO using our Z-80's LDIR instruction when it is empty. All we need remember is that TIME itself is finite. Or is it indeed? If we speed up our TRS-80's clock 50% or even 100%, Bryan Mumford is working on the latter now, we can double the throughput of our TRS-80 with the 'flick' of a switch. After all, the Z-80A is guaranteed to work with a 4 MHz clock, and here we are loafing along at 1.77 MHz (or the 2.66 MHz clock I am using RIGHT NOW to write this page with Electric Pencil). Time is only relatively finite depending upon your viewpoint.

LPRINT WHILE 'A' REGISTER INPUTS TO SIMULATED FIFO:

This little demonstration program whose source code with comments is on page 40 and object code on page 41 may be entered in about 8 minutes with Radio Shack's excellent Editor/Assembler if you can do without the comments. So, give it a try. It is similar to our earlier program in BASIC with a few extra convenient features that you may find useful. They include:

- Start LPRINT by pressing <u>SHIFT</u>/RIGHT ARROW.

- Stop LPRINT by pressing <u>SHIFT</u>/LEFT ARROW.

- When full video display has been LPRINTed auto CLS.

- And, auto turn-off LPRINT after full video LPRINTed.

Notice that the author mixes up hex/decimal addresses and data throughout the program. It really makes no nevermind as our Editor/Assembler could care less just as long as we specify exactly what format we are using. By all means use that which is easiest for YOU and don't worry about it. Maybe we should hang a sign on the window saying: "Any dialect understood by the EDTASM is spoken here and ok, 79 - 75, 4FH - 4BH."

```
00100 ; LPRINT WHILE 'A' REGISTER INPUTS TO SIMULATED FIFO
00110         ORG     7D00H         ;=32000 DECIMAL
00120         CALL    01C9H         ;CLS
00130 PRINT   DEFB    0             ;LPRINT SIGNPOST 1=LPRINT
00140 CARRET  DEFB    0             ;LPRINT CHAR/LINE COUNTER
00150 LINES   DEFB    0             ;VIDEO LINES/PAGE COUNTER
00160         LD      IY,15360      ;VIDEO LOCATION COUNTER
00170 KYBD    CALL    002BH         ;KEYBOARD SCAN ROUTINE
00180         CP      25            ;SUBTRACT 25=RIGHT ARROW
00190         JR      Z,SET         ;GOTO SET IF ZERO
00200         CP      24            ;SUBTRACT 24=LEFT ARROW
00210         JR      Z,RESET       ;GOTO RESET IF ZERO
00220         CP      0             ;0 = NO KYBD CHARACTER
00230         JR      NZ,VIDEO      ;GOTO VIDEO IF CHARACTER
00240         LD      A,(PRINT)     ;+1 = LPRINT OUTPUT
00250         CP      1             ;SUBTRACT 1
00260         JR      Z,READY       ;GOTO READY IF ZERO
00270         JR      KYBD          ;RETURN TO KEYBOARD
00280 VIDEO   CALL    033H          ;OUTPUT 'A' TO VIDEO
00290         JR      KYBD          ;RETURN TO KEYBOARD
00300 SET     LD      A,1           ;+1 TO 'A' REGISTER
00310         LD      (PRINT),A     ;LOAD LPRINT SIGNPOST
00320 READY   LD      A,(37E8H)     ;LPRINT HANDSHAKE MEM
00330         CP      63            ;SUBTRACT 63
00340         JR      NZ,KYBD       ;GOTO KYBD IF NOT READY
00350         LD      A,(IY)        ;NEXT VIDEO CHARACTER
00360         CALL    003BH         ;LPRINT CHARACTER IN 'A'
00370         INC     IY            ;+1 TO VIDEO LOCATION
00380         LD      A,(CARRET)    ;LPRINT CHAR. COUNTER
00390         INC     A             ;+1 TO CHARACTER COUNTER
00400         CP      64            ;SUBTRACT 64
00410         JR      Z,RETURN      ;GOTO RETURN IF 64 CHARS.
00420         LD      (CARRET),A    ;STASH CHAR/LINE IN MEM
00430         JR      KYBD          ;RETURN TO KEYBOARD
00440 RESET   LD      A,0           ;0 TO 'A' REGISTER
00450         LD      (PRINT),A     ;TO 'STOP' LPRINT
00460         JR      KYBD          ;RETURN TO KEYBOARD
00470 RETURN  LD      A,0           ;ZERO TO 'A' REGISTER
00480         LD      (CARRET),A    ;RESET CHAR/LINE COUNTER
00490         LD      A,(37E8H)     ;LPRINT STATUS LOCATION
00500         CP      63            ;63 = LP READY HANDSHAKE
00510         JR      NZ,RETURN     ;TO RETURN IF NOT READY
00520         LD      A,0DH         ;0DH = CARRIAGE RETURN
00530         CALL    003BH         ;LP DRIVER MEM - DO IT
00540         LD      A,(LINES)     ;NUMBER LINES LPRINTED
00550         INC     A             ;ADD 1 TO 'A' REGISTER
00560         CP      16            ;SUBTRACT 16
00570         JR      Z,CLS         ;GOTO CLS IF ZERO
00580         LD      (LINES),A     ;UPDATE LINE COUNTER
00590         JR      KYBD          ;RETURN TO KEYBOARD
00600 CLS     LD      IY,15360      ;RESET VIDEO MEM COUNTER
00610         LD      A,0           ;ZERO TO 'A' REGISTER
00620         LD      (LINES),A     ;RESET LINE COUNTER
00630         LD      (PRINT),A     ;TURN OFF LPRINTER
00640         CALL    01C9H         ;CLS FOR NEW PAGE VIDEO
00650         JR      KYBD          ;RETURN TO KYBD
00660         END     7D00H         ;EL FIN = EL BEGUINE
```

```
7D00              00110          ORG     7D00H
7D00 CDC901       00120          CALL    01C9H
7D03 00           00130 PRINT    DEFB    0
7D04 00           00140 CARRET   DEFB    0
7D05 00           00150 LINES    DEFB    0
7D06 FD21003C     00160          LD      IY,15360
7D0A CD2B00       00170 KYBD     CALL    002BH
7D0D FE19         00180          CP      25
7D0F 2816         00190          JR      Z,SET
7D11 FE18         00200          CP      24
7D13 2833         00210          JR      Z,RESET
7D15 FE00         00220          CP      0
7D17 2009         00230          JR      NZ,VIDEO
7D19 3A037D       00240          LD      A,(PRINT)
7D1C FE01         00250          CP      1
7D1E 280C         00260          JR      Z,READY
7D20 18E8         00270          JR      KYBD
7D22 CD3300       00280 VIDEO    CALL    033H
7D25 18E3         00290          JR      KYBD
7D27 3E01         00300 SET      LD      A,1
7D29 32037D       00310          LD      (PRINT),A
7D2C 3AE837       00320 READY    LD      A,(37E8H)
7D2F FE3F         00330          CP      63
7D31 20D7         00340          JR      NZ,KYBD
7D33 FD7E00       00350          LD      A,(IY)
7D36 CD3B00       00360          CALL    003BH
7D39 FD23         00370          INC     IY
7D3B 3A047D       00380          LD      A,(CARRET)
7D3E 3C           00390          INC     A
7D3F FE40         00400          CP      64
7D41 280C         00410          JR      Z,RETURN
7D43 32047D       00420          LD      (CARRET),A
7D46 18C2         00430          JR      KYBD
7D48 3E00         00440 RESET    LD      A,0
7D4A 32037D       00450          LD      (PRINT),A
7D4D 18BB         00460          JR      KYBD
7D4F 3E00         00470 RETURN   LD      A,0
7D51 32047D       00480          LD      (CARRET),A
7D54 3AE837       00490          LD      A,(37E8H)
7D57 FE3F         00500          CP      63
7D59 20F4         00510          JR      NZ,RETURN
7D5B 3E0D         00520          LD      A,0DH
7D5D CD3B00       00530          CALL    003BH
7D60 3A057D       00540          LD      A,(LINES)
7D63 3C           00550          INC     A
7D64 FE10         00560          CP      16
7D66 2805         00570          JR      Z,CLS
7D68 32057D       00580          LD      (LINES),A
7D6B 189D         00590          JR      KYBD
7D6D FD21003C     00600 CLS      LD      IY,15360
7D71 3E00         00610          LD      A,0
7D73 32057D       00620          LD      (LINES),A
7D76 32037D       00630          LD      (PRINT),A
7D79 CDC901       00640          CALL    01C9H
7D7C 188C         00650          JR      KYBD
7D00              00660          END     7D00H
00000 TOTAL ERRORS
```

LPRINT WHILE 'A' REGISTER INPUTS TO SIMULATED FIFO REVIEW:

Let's take a brief run through this little demonstration program. The few extra 'bells and whistles,' as mentioned earlier, may give you some ideas when writing a program for your own special applications. The program may be located most anywhere in FREE memory. Just change lines 110 and 660 as appropriate. Since this program only utilizes 125 bytes of MEM, you may easily sneak it into MEM well BENEATH any disk BASIC program beginning at 26810 where disk BASIC programs begin. Virtually, FREEBIE memory space. There is lots and lots available here and elsewhere that may be used judiciously if you are careful. Take a look at your MEM from 24000 to 26810 using either just plain 'ole PEEK or the disassembler from the last Chapter, BUT be sure to TEST this same supposedly FREE memory AFTER you enter a number of strings and so forth that MAY use this area for pointers, etc. Microsoft, Randy Cook, and Apparat did not leave nearly 1500+ FREE bytes between 17129 and 26810 unused, unless they didn't know any better. We both believe and hope they KNEW better, so be careful when diddling around with ALL this empty MEM space.

LINE 130:
A byte saved here in MEM will be used as our LPRINT signpost. A zero here means NO line printer output and a +1 = LPRINT.

LINE 140:
A byte saved here in MEM will be used for our LPRINTer's characters per line counter. At 64, we will issue a carriage return and reset this byte to zero.

LINE 150:
A byte saved here in MEM will be our video lines per page counter for the LPRINTer. After printing 16 lines we will CLS, turn-off the line printer, reset this byte to zero, and reset the video MEM counter in the next line to 15360.

LINE 160:
Is a SNEAKY but useful way of saving MEM by using the Z-80's IY index register for keeping track of our video MEM location for LPRINT output. The IY register is never used by Level II or disk BASIC so we might as well put it to good use here and save 2 bytes of MEM.

LINES 170 - 270:
First scans the keyboard for instructions in line 170. If NO key is pressed and PRINT still contains a zero (no LPRINT yet desired), the zero in the 'A' register falls through to line 270 which sends the program back to line 170 for another keyboard scan.

LINES 180 - 190:
Test the 'A' register to see IF you pressed the SHIFT-RIGHT ARROW keys to start LPRINTing. IF so, it jumps to lines 300 & 310 that stash away a +1 in the PRINT memory location.

LINES 200 - 210:
Test the 'A' register to see if you pressed SHIFT-LEFT ARROW to stop the line printer and IF so, jump off to lines 440-460 to stash away a zero at PRINT's MEM location and then returns for another keyboard scan.

LINES 220 - 230:
Zap the program off to VIDEO to output in line 280 IF you have pressed a key on the keyboard other than SHIFT-RIGHT or LEFT ARROW. All video display commands are available except CLEAR and BREAK. Line 290 then returns for another keyboard scan.

LINES 240 - 260:
Are really our SPOOLING (or whatever you wish to call it) work horses in this program. IF you have previously instructed the program to commence LPRINTing, anytime the keyboard output to the 'A' register is zero during a keyboard scan, the program falls through the previous compares and JR's to line 260 which sends the program off to READY in line 320, SINCE THE COMPUTER IS IDLING and has nothing else to do.

LINES 320 - 340:
First tests 'printer ready' at 37E8H to see if a 63 = ASCII ? = "what's next? send me the next character to LPRINT," or = 255 = "I'm busy." IF busy, line 340 sends the program back for another keyboard scan, or IF READY it falls through to the next line.

LINES 350 - 360:
Load the next character to be LPRINTed from the video MEM location contained in the IY index register and LPRINTs it in line 360 via CALLing 003BH. This CALL is a new one for readers of Vols. 1 & 2, so ADD it to Vol. 1's page 33 and store it in your own memory. It is much FASTER and EASIER to use than CALL 032AH since we first have to load 409CH with a +1 to direct its output to LPRINT. A 'thank you' to Bryan Mumford for this improvement.

LINES 370 - 430:
Increment our characters per line LPRINT counter at CARRET's MEM location, checks to see if a 64 character line is complete, and IF so, jumps off to RETURN. IF the 64 character line is NOT complete, line 430 sends the program back for another keyboard scan.

LINES 470 - 590:
Reset our characters per line counter to zero, issue a carriage return in line 530 WHEN the printer is ready for it, increments our lines per video page by +1 in line 550, and jumps off to CLS if we have completed printing a video page.

LINES 600 - 650:
Reset the IY index register to the 1st video MEM location at 15360, reset LINEs per video page to zero, stops LPRINT output by stashing a zero at PRINT, and then CLS for a new page of video.

EVERYTHING TAKES A FINITE AMOUNT OF TIME - REALLY:

You bet it does. When you compare the two 'sort of' SPOOLING programs in this Chapter, you can easily see the difference. In our assembly language program, we say again, the line printer doesn't even know anything else is going on. By speaking our Z-80's own native language, a rare dialect of 10th century FAGGIN (from Fagginslyvania - a remote province on the Swiss-Italian border in the Alps), our program runs a good 300+ times faster than through our BASIC interpreter. In later Chapters we will put all this spare time to much better use than just running a 'spoon and egg' race between you, the keyboard, and your line printer.

SOME FOOD FOR THOUGHT:

Let's say we had a separate FIFO capable of clocking data out at a 15 MHz rate that we filled at normal TRS-80 speed. When our FIFO was FULL, let us clock it out at a 15 MHz rate and use this data to fm (frequency modulate) a 10,000 MHz (10 GHz) Gunnplexer microwave transmitter. Would it work? Sure it would. This is called "burst" modulation. If we enciphered a few bytes of each burst to tell the receiver our NEXT exact transmission frequency, we could FREQUENCY HOP all over the band totally secure from interception.....this is a much simplified version of how D.O.D. satellite communications will be made during the 1980s.

Actually, German submarines used a modulation format somewhat similar to this during World War II on the high frequency bands to communicate with the fatherland, but (there's always a but) International Telephone & Telegraph's electronic cathode ray tube direction finder, invented by Dr. Henri Busignies (a refugee from German occupied France), was able to "latch" onto these short bursts, and off went the patrol bombers with depth charages (every lock has a key). Gud luk U-boat Commanders.

SUMMARY:

Both of the programs in this Chapter are intended only as 'openers' to the SPOOLING/FIFO game. The number of modifications you may make to the assembly language program to suit your own purposes are limited only by your imagination. Your line printer could just as well be sequentially printing out the 5 to 20 pages you stored in MEM with Volume 2's Chapter 7 program, or the RECEIVE data stored in MEM from Volume 2's Chapter 8 program, WHILE video displays the new incoming data from a computer bulletin board system, or what-have-you. YOU could then select whatever you wanted LPRINTed by storing it with the '123' keys, and let that incoming data you did not wish to LPRINT just scroll off the screen into never-never land or wherever unwanted bytes find their resting place.

Now, let's have at the easy questions for this Chapter and then continue our dialogue at your pleasure and convenience.

- CHAPTER 4 -

AN INTRODUCTION TO TRS-80 INTERRUPTS

INTRO TO AN INTRODUCTION:
Interrupt handling is a fascinating aspect of microprocessor programming. Without extensive hardware modification, we have two forms of interrupts available to us on the TRS-80; the non-maskable interrupt which is activated by the RESET button on the rear of the keyboard, and the maskable interrupt MODE ONE which this Chapter is all about. There is nothing really mysterious about the words non-maskable and maskable. All they mean is that a non-maskable interrupt CANNOT be turned off, and a maskable interrupt may be ENABLED or DISABLED as desired with the EI or DI TRS-80 assembler instructions. This Chapter will cover a few simple hybrid BASIC/ASSEMBLY language programs to illustrate how the interrupt MODE ONE instruction can be used and will hopefully lead you into developing your own more sophisticated interrupt subroutines for whatever purpose suits your fancy. By a hybrid program we mean one that uses both assembly language and BASIC to illustrate the points covered. We have chosen this format primarily to enable those readers that do not have an Editor/Assembler to quite simply load the object code part of the program with BASIC Read-Data statements. The only hardware required in this Chapter is a small, normally-open pushbutton switch, 3 for 99 cents at Radio Shack, and two wires connecting the switch to the Z-80's maskable interrupt pin and ground. These connections may be made to the connector on the rear of the TRS-80 keyboard via pins 21 and 39 respectively, if you have a spare connector ($4.50 from Hobbyworld), or may simply be tack soldered to the keyboard's printed circuit board JUST INSIDE of the connector fingers. Figure 4-1 is a rear view of the keyboard's 40 connector fingers (viewed from the outside):

```
                         - TOP -

 1  3  5  7  9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39
 *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *
 ___________________________________________________________
 *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *
 2  4  6  8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40

                       - BOTTOM -
```

Figure 4-1

Pins 8, 29, & 37 = signal ground. Pin 21 = maskable interrupt. Pin 21 is normally held at a logic 1 = approximately +5 volts dc by the regulated +5 volt power supply and is fed through resistor R40, a 4.7K ohm 1/4 watt resistor. Let's take a brief look at the Z-80's interrupt capabilities and then see how very easy it is to implement one of them on the TRS-80 without cobbling up our pride and joy by cutting traces hither and yon, and adding all sorts of integrated cicruits to make it work properly.

Z-80 INTERRUPT FLIP FLOPS 1 AND 2 (IFF1 & IFF2):
Here are the "stop" and "go" traffic lights for the Z-80 interrupt system. IFF1 stores the enable interrupt (EI) status or disable interrupt (DI) status of our program for the maskable interrupts. Whenever a non-maskable interrupt is received we certainly DO NOT want a maskable interrupt (lower priority) to be able take charge, hence IFF1's on or off status is stored in IFF2 and IFF1 is turned "off" = maskable interrupts disabled while the non-maskable interupt does its thing.

This is mentioned only for you theorists since the TRS-80 uses the non-maskable interrupt (NMI) to effect a system reset whenever our TRS-80 gets thoroughly bolixed up, and we have to press the RESET switch on the back of the keyboard.

NON-MASKABLE INTERRUPT:
In some Z-80 based microcomputers, the non-maskable interrupt (NMI) function, Z-80 pin 17, is used exclusively to sense a catastrophic power failure. Whenever pin 17 goes low, we'll call low < 1.7 volts dc and high > 2.2 volts, the Z-80 senses the NMI low at the end of the instruction cycle (it checks for interrupts every instruction cycle), saves the status of IFF1 in IFF2, turns IFF1 "off" so that any maskable interrupts will be ignored, saves the value of the program counter in the stack, and then charges off to memory location 066 hex to do its rescue thing before ALL power is lost, memory and program wiped out, and maybe zillions of bytes lost forever. In this theoretical microcomputer let's assume the VERY FIRST THING memory location 066 hex does is "turn on" backup battery power via an OUT port instruction and then proceeds with a power down group of instructions to conserve power until normal power is restored and NMI brought back up to > 2.2 volts. At the end of this emergency power down subroutine we would find the Z-80 instruction RETN which = pop the saved address off the top of the stack (return from non-maskable interrupt) and return to our regular program WHEN normal power is restored. RETN also restores IFF1 to its on or off condition that existed BEFORE the NMI.

RETurning to the real world of our TRS-80, the RESET switch on the back of the keyboard that brings the NMI low, sends our Z-80 off to 066 hex where it effectively reinitializes the system. MEM location 069H checks to see whether we have the expansion interface (and disks) installed and reinitializes via that route, or if NOT, sends us back to MEM 0000H just like we had first turned the power on. Though outside the rules of this particular "ballgame," you intrepid experimenters should find it a not too difficult challenge to modify the input to Z-80 NMI pin 17 so that it senses an incipient power failure (forinstance < 4.5 volts dc), and via an OUT port instruction turns on a nicad backup battery power supply. You must intercept the DOS intialization routine to do so.

Z-80 MASKABLE INTERRUPTS:

The Z-80 microprocessor offers the system designer THREE distinctly different types of maskable interrupts. Compared to the old 8080 microprocessor that only had one, this is a giant step forward. Let's take a brief look at each one and then zero in on the one that we may implement on the TRS-80 with only a switch.....or just a single bare wire to ground pin 21 on the 40 pin keyboard connector.

Both non-maskable and maskable interrupts (if IFF1 is enabled) are recognized by the Z-80 during the last "T" state of every instruction's last machine cycle if the BUSRQ (bus request) line on the Z-80 (pin 25) has NOT been brought low. BUSRQ is tied to +5 volts dc on the TRS-80 so we may forget about it except during service testing. As previously mentioned, the non-maskable interrupt has the first priority, and if NMI has been brought low, automatically disables IFF1 and stores it in IFF2. If maskable INT has been brought low and IFF1 enabled, then off it goes into the maskable interrupt mode that has been set, IM 0, IM 1, or IM 2. The Z-80 initializes itself automatically in mode IM 0, so if we wish to use another mode, we must tell it so.

In the following discussion let us assume that the appropriate interrupt mode has been set and interrupt enabled (EI to enable IFF1) on our theoretical Z-80 based microcomputer. Let's see how each mode does its thing.

INTERRUPT MODE 0:

1. Z-80 INT pin 21 goes low = interrupt has occured.
2. Interrupt recognized by Z-80 at end of current instruction AFTER Z-80 INT pin 21 has gone LOW.
3. IFF1 turned off = disable interrupts, plus 2 WAIT states.
4. When both Z-80 M1 (pin 27) and IORQ (pin 20) have gone low, this generates an interrupt acknowledge signal to tell an external interrupt controller "I'm ready for an RST interrupt vector."
5. External interrupt controller outputs an RST instruction to DATA bus. This may be RST 0H, RST 8H, RST 10H, RST 18H RST 20H, RST 28H, RST 30H, or RST 38H. Only the 3rd, 4th, and 5th most significant bits of the 8 bit object code instruction are necessary as all the rest are always 1's.
6. After strobing in the data, the Z-80 now recognizes the instruction and saves the contents of the program counter in the stack so that when the interrupt subroutine is completed, an RETI instruction is all that is necessary to resume the normal program at the EXACT place in MEM where the interrupt occured.
7. The RST XXH instruction is now executed which takes us to page one in MEM where the appropriate RST subroutine does its thing, whatever it may.
8. Depending upon how many registers the particular interrupt subroutine uses, it is a MUST to save their values either in the stack or by swapping alternate registers pairs, if only AF, BC, DE, and HL register pairs are used.

9. At the end of the interrupt subroutine we should return the values of those register pairs that were saved, and then enable interrupts with the EI instruction, before using the RETI instruction to resume the normal program.

What if an INTERRUPT occurs AFTER the EI instruction and BEFORE the RETI instruction.....what happens?

You are awake again, Gridley. A good question. The Faggin-sylvanians who designed this remarkable machine thought about that one too. The EI instruction does not reset IFF1 UNTIL after the NEXT instruction has been completed, usually a return or return from interrupts. It takes real effort to bolix up this wonderful Z-80 microprocessor.

10. IFF1 is enabled AFTER the completion of the next instruction following the EI instruction. Also, if we used an RETI instruction to return to the normal program, the external interrupt controller would receive a signal saying, "interrupt completed, go back to sleep, reset whatever gizmos you must, and await the next bugle call = INT low."

INTERRUPT MODE 1:

1. Z-80 INT pin 21 goes low = interrupt has occured.
2. Interrupt recognized by Z-80 at end of current instruction cycle.
3. IFF1 turned off = disable interrupts, plus 2 WAIT states.
4. Interrupt acknowledge cycle is IGNORED.
5. The contents of the program counter are automatically saved in the stack.
6. Z-80 issues an RST 38H instruction and jumps to 38H in MEM to do its interrupt subroutine thing.
7. Again, we MUST save all appropriate register values to not foul-up the normal program.
8. Upon completion of IM 1 subroutine, restore appropriate register values, IFF1 should be enabled (EI), and an ordinary RET instruction executed to POP the program counter address off the top of the stack to allow normal program resumption.

This is the interrupt mode we will use later in this Chapter. There is absolutely nothing SNEAKY or TRICKY about using it IF one remembers to save the value of EVERY POSSIBLE memory location that BOTH the normal program and interrupt subroutine utilizes.

INTERRUPT MODE 2:

This is the most versatile and useful of all three Z-80 interrupt modes as it uses the Z-80's "I" index register PLUS the interrupt controller's output to the DATA bus to form an address that can provide a full page of 128 interrupt service

vectors to take the program most anywhere in memory desired. Let's see how IM 2 would work IF we had the external interrupt controller and IF we chose to cobble up the expansion interface to allow it to work.

1. Z-80 INT pin 21 is taken low = interrupt has occured.
2. Interrupt recognized by Z-80 at end of current instruction cycle.
3. IFF1 turned off=disable interrupts, plus 2 WAIT states.
4. Stores program counter in RAM stack.
5. Forms a 16 bit address consisting of 8 most significant bits from the "I" register and 8 least significant bits (last one always a zero) from the interrupt controller.
6. Loads value at this 16 bit address and address +1 into program counter.
7. JUMPS to this address.
8. Executes interrupt subroutine beginning at this address.
9. Interrupt subroutine should save any and all registers and/or MEM locations commonly used by regular AND interrupt programs BEFORE proceeding with interrupt subroutine.
10. Upon completion of interrupt subroutine, restore all registers and/or commonly used MEM locations, then enable interrupts and RETI to regular program.
11. RETI tells the interrupt controller to reintialize itself and go back to sleep till the next INT, as well as POP the stack with the return address to the regular program.

This is a neat scheme with great merit for those special applications that require numerous different interrupt programs, such as are used by industrial control systems in chemical engineering applications; i.e., where multi-step processing is required. With proper programming, a little 8 bit Z-80 based micro can often replace PDP-11 and IBM 360 minis and macros in numerous process control applications.

The IM 2 mode does require 19 full clock cycles to fully implement itself.....and a table of vectors to be loaded into RAM. Also, the "I" register must loaded from the "A" register with the page number (8 most significant bits of the address) of the address table. This is a small price to pay for the flexibility it offers.

Now let's JP back to the real world of the TRS-80 and its simple interrupt MODE 1 programs we can easily load and run without buying external interrupt controllers such as PIO's and SIO's and hacksawing up our precious expansion interface. To keep from fouling-up our sort-of realtime clock in the expansion interface which does DOUBLE DUTY by serving the floppy disk controller as well as providing us with a somewhat clock-like 25 millisecond "heart beat," we will TURN-OFF the expansion interface and run our TRS-80 just like a plain 'ole' 16K non-disk machine. There is NO need to disconnect the keyboard to expansion interface connector while running any of the following programs. Just make sure the expansion interface on-off switch is OUT = OFF. Let's get to work now.

HOOKING UP THE INTERRUPT SWITCH TO THE TRS-80 KEYBOARD:

First, we need some sort of normally-open push button switch with which to activate the Z-80's ILT pin 16, by grounding it. Since Z-80 pin 16 is very conveniently brought out on pin 21 of the TRS-80 rear keyboard connector, we have two choices. IF you are adventurous and used to soldering #22 insulated wire to very small traces on printed circuit boards, by all means disassemble your keyboard and solder two 6 inch long wires to the traces BEHIND pin 21 (INT) and pin 39 (ground), on the 40 pin keyboard, rear connector. Bring these leads out along the side of the 40 pin connector and solder them to a Radio Shack #275-1547 normally open, single pole-single throw, pushbutton switch. ANY normally open, pushbutton switch in the junk box will work just as well.

IF you do not wish to void your TRS-80 warranty by opening the keyboard assembly, a 40 pin TRS-80 edge connector, #1980, may be ordered from: Hobbyworld Electronics, 19511 Business Center Drive, Northridge, California, for $4.75 plus postage. Simply solder the leads from the pushbutton switch to the connector's pins 21 & 39. Removing the expansion interface connector from the keyboard connector will make no nevermind, as we will not be using the expansion interface with the following interrupt demonstration programs to avoid modifying the "clock's" wiring and printed circuit board traces.

TRS-80 INTERRUPT DEMONSTRATION PROGRAMS:

There are all sorts of ways to approach this subject. The one we like best to start with may be loaded with your Editor/ Assembler OR via a simple BASIC read-data mini-program. Figure 4-2 is the source code, Figure 4-3 is the object code, and Figure 4-4 the BASIC read-data program to POKE the object code into MEM if you do not have an Editor/Assembler.

```
00100 ; INTERRUPT DEMONSTRATION PROGRAM 1 - SOURCE CODE
00110
00120          ORG     32000          ;START PROGRAM HERE
00130          PUSH    HL             ;SAVE HL VALUE IN STACK
00140          LD      HL,(32100)     ;COUNTER STASH IN MEMORY
00150          INC     HL             ;ADD +1 TO HL REGISTERS
00160          LD      (32100),HL     ;STASH IT AWAY IN MEMORY
00170          POP     HL             ;RESTORE HL FROM STACK
00180          EI                     ;RE-ENABLE INTERRUPTS
00190          RET                    ;RETURN ADDRESS FM STACK
00200          ORG     32200          ;MOVE PROGRAM TO 32200
00210          IM      1              ;SET INTERRUPT MODE ONE
00220          EI                     ;ENABLE INTERRUPTS
00230          JP      114            ;RETURN TO BASIC 'READY'
00240          END     32200          ;INITIALIZE AT 32200
```

Figure 4-2

INTERRUPT DEMONSTRATION PROGRAM 1 - OBJECT CODE

```
7D00            00120         ORG       32000
7D00 E5         00130         PUSH      HL
7D01 2A647D     00140         LD        HL,(32100)
7D04 23         00150         INC       HL
7D05 22647D     00160         LD        (32100),HL
7D08 E3         00170         POP       HL
7D09 FB         00180         EI
7D0A C9         00190         RET
7DC8            00200         ORG       32200
7DC8 ED56       00210         IM        1
7DCA FB         00220         EI
7DCB C37200     00230         JP        114
7DC8            00240         END       32200
```

Figure 4-3

```
4 'READ-DATA SUBSTITUTE FOR ASSEMBLY LANGUAGE PGM
6 '
10 FORX=32000TO32010:READM:POKEX,M:NEXT
20 DATA 229,42,100,125,35,34,100,125,225,251,201
30 FORX=32200TO32205:READM:POKEX,M:NEXT
40 DATA 237,86,251,195,114,0
50 END
```

Figure 4-4

All the BASIC Read-Data program in Figure 4-4 is doing, is to POKE the identical values in MEM that Figure 4-3's object code loads into MEM. Earlier we said, this would be a HYBRID program using both BASIC and assembly language to KISS (keep it simple, Simon), as the worthy poet, David Lein-W6OVP often advises. Figure 4-5 is the BASIC part of our hybrid program.

```
4 'BASIC PART OF HYBRID INTERRUPT DEMO PGM 1
6 '
10 DEFINTY:CLS:POKE32100,0:POKE32101,0
20 POKE16402,195:POKE16403,0:POKE16404,125
30 PRINT@170,PEEK(32100)+256*PEEK(32101)
40 Y=Y+1:PRINT@298,Y:GOTO30
```

Figure 4-5

WHAT'S GOING ON HERE ? ? ?

Well Gridley, you are getting two programs for the price of one. That is what's going on here. Seriously, let's look at Figure 4-2's source code program first, and then we'll JP to Figure 4-5's BASIC program to see how they work together. The two programs are OBVIOUS to everyone else, Gridley, but for your benefit we'll take them both line by line. Remember to load them both on cassette as the expansion interface, if you have one, IS TURNED OFF.

GENERAL PROGRAM CONCEPT:

Before jumping into each program line 'head first,' here is the general concept of what we are trying to do.

1. Set-up a simple BASIC program that includes a counter and display the counter's value on video.
2. Anytime an INTERRUPT is received, leave the BASIC program and jump to a machine language subroutine in high MEM that also has a counter and will continue counting AS LONG AS THE INTERRUPT switch is closed (INT grounded).
3. Whenever the INTERRUPT switch returns to normal, is open, the machine language program will RETurn control to the BASIC program WHEREVER it was interrupted.
4. To KISS (keep it simple), we will use the BASIC program to display both the BASIC as well as the interrupt counter's values. As such, we do not have SAVE all the registers, the ACCUM's values, cursor position, etc., etc. and everything including the kitchen sink. We will do that later.
5. The tail end of the short machine language subroutine will intialize the Z-80 in INTERRUPT mode one, enable the interrupt function, and then return to BASIC 'READY' whenever we load the SYSTEM program and type /32200 after it is loaded.

FIGURES 4-2 & 4-3 SOURCE/OBJECT CODES PROGRAM 1:

Since our Editor/Assembler will work with decimal just as easily as hex, let's use decimal for the time being for the sake of clarity. IF you have 8 fingers on each hand, by all means use hex if that suits your fancy.....Gridley is counting his fingers.

We will start our object code subroutine at 32000 in MEM to keep it up in high memory, well out of the way of any lengthy BASIC program you might wish to use. The only register we will be using is HL. As such, line 130 saves it in the stack so as not to foul up any type of program, BASIC or otherwise, that our INTERRUPTED PROGRAM will be coming FROM. The EXX instruction would have done the job just as well.....actually better since it is both faster and does not use stack MEM. This simple subroutine is nothing more than a counter that increments the HL register pair by +1, WHENEVER the Z-80 senses a low at its INTERRUPT input.

Line 140: MEM locations 32100 and 32101 hold the value of our INTERRUPT counter. This line loads that value into the HL register pair.

Line 150: Increments the HL register pair by +1.

Line 160: Stashes away the incremented value into 32100/01.

Line 170: Restores the original value of HL from the stack.

Line 180: Re-enables the interrupt function (IFF1) as it was turned off automatically when the INT was first acknowledged.

Line 190: Pops the RETurn address of our interrupted BASIC program off the top of the stack.

Line 200: This line tells our assembler to move the rest of this assembly language subroutine to 32200 in MEM so that we may intialize it by typing in /32200, then ENTER.

Line 210: Puts our Z-80 into interrupt mode 1. Remember, it was automatically put in IM zero when first turned-on.

Line 220: Enables the interrupt function via interrupt flipflop IFF1. You will recall from Volume 2, that the number 0000 instruction in ROM was DI, disable interrupts. Therefore, we have to 'turn it on.'

Line 230: Takes us back to basic's READY when we intialize.

Line 240: Tells our assembler THE END and POKEs 32200 into RAM after the SYSTEM command has loaded this mini-subroutine, so we DO NOT HAVE TO USE the /32200, then ENTER the first time. Just typing in a / then ENTER will usually do it......BUT, it is a good idea to remember /32200 then ENTER, and get used to using it to avoid pratfalls as this RAM MEM location may be easily changed by an unfriendly ghost that has been known to drop bat guano EVEN into hermetically sealed keyboard keys.

HOW DO IT DO IT ? ? ?

Not easily Gridley, but to be forewarned, is to be forarmed.

FIGURE 4-5 BASIC PART OF HYBRID INTERRUPT DEMO PGM 1:

Line 10 first defines variable 'Y' as an integer to allow the counter to run as fast as possible and then CLS, plus zeroing out our interrupt counter at MEM locations 32100 and 32101. It would have been just as easy to zero them out in the assembly language program. Either way is ok.

Line 20: In standard Level II non-disk BASIC, an RST 38H sends the program off to MEM location 16402 which contains an EI, enable interrupt instruction and 16403 which contains a RETurn instruction. This line POKEs a JP into 16402 and the address of 32000 decimal into 16403 and 16404. Like line 10, it would have been just as easy to have the assembly language program load the JP address into these MEM locations. Either way ok.

Line 30: Converts our interrupt counter MEM locations 32100 and 32101 to an integer covering the range of zero to 65535. Again, the reason we are having BASIC do the hard work in these hybrid programs is to keep from having to SAVE everything but the kitchen sink in either the stack or MEM as we switch back and forth between our BASIC program's counter and the INTERRUPT subroutine's counter.

Line 40: Is our inordinately simple BASIC counter.

LOADING THE HYBRID INTERRUPT DEMONSTRATION PROGRAMS:

1. REMEMBER, the expansion interface is turned OFF to keep the 'real time' clock from mucking about with interrupts and RST 38Hs.

2. It makes no nevermind whether you load the SYSTEM or BASIC program first. Either way is ok, but do not forget the /32200 after the SYSTEM program has loaded to both set IM 1 and enable interrupts.

3. If you used the BASIC Read-Data program in Figure 4-4 to load the interrupt subroutine, you must still type in SYSTEM and then ENTER, followed by /32200 and then ENTER.

4. After both programs are loaded and you RUN, the video display will show a '0' above the BASIC program's counter UNTIL you close the little INTERRUPT pushbutton switch.

WHEW....the machine language counter really moves, doesn't it? Somewhere in the vicinity of 20,000 counts for every second you have held the interrupt switch closed. By all means see how briefly you can close your interrupt switch; i.e., what is the minimum count you can close it and open it manually? ? ?

Well, so much for our first INTERRUPT experiment. It was about as simple as we could dream up and yet still illustrate the Z-80's INTERRUPT mode one within the constraints of our TRS-80 environment. Hopefully, it gave you some 'food for thought' regarding ways and means of utilizing the Z-80's interrupt function. It is doubtful if you will use it for chemical processing or paper-tape-punch SPOOLING. Its applications are limited only by your imagination and your particular requirements. Our first USEFUL application of using the interrupt function was to "burst" load an 8 bit wide by 320 bytes deep first-in-first-out, FIFO, buffer that drove a little solid-state Morse code transmitting system described in the December 1975 issue of QST magazine. Whenever the FIFO empty flag was raised, the Z-80's interrupt pin was taken low, and the interrupt subroutine "burst" loaded the FIFO from video memory, cleared those 320 bytes from the TRS-80 video display, and then returned to the normal keyboard input to video display program. The purpose of this exercise was to allow "type ahead" capability while transmitting Morse code at a constant rate. Though not very sophisticated, to say the least, it worked quite well.

Let's modify the two programs in Figures 4-3 and 4-5 to count interrupts up to 65535 X 255 = 16,711,425 by adding another MEM stash location and a compare, CP, to know when to load it. There is virtually no reasonable limit to the maximum number of interrupts you may count by cascading MEM locations and compares, if you wish. Yet another MEM stash would allow:

16,711,425 X 255 = 4,261,413,375 interrupt counts if desired.

```
00100 ; INTERRUPT DEMONSTRATION PROGRAM 2  -  SOURCE CODE
00110
00120          ORG     32000          ;START PROGRAM HERE
00130          EX      AF,AF'         ;SWAP ALT. AF REGISTERS
00140          EXX                    ;SWAP ALT. BC-DE-HL REGS.
00150          LD      HL,(32100)     ;32100/01 = COUNTER STASH
00160          INC     HL             ;ADD +1 TO HL REGISTERS
00170          LD      (32100),HL     ;STASH IT AWAY IN MEM
00180          LD      A,H            ;LOAD H INTO A REGISTER
00190          CP      255            ;SUBTRACT 255 & SET FLAGS
00200          JP      Z,COUNT        ;IF ZERO, GOTO COUNT
00210 FINIS    EX      AF,AF'         ;RESTORE AF ORIG. VALUES
00220          EXX                    ;RDSTORE BC-DE-HL VALUES
00230          EI                     ;ENABLE INTERRUPTS
00240          RET                    ;POP BASIC'S RET ADDRESS
00250          ORG     32200          ;MOVE PROGRAM TO 32200
00260          IM      1              ;SET INTERRUPT MODE ONE
00270          EI                     ;SET IFF1 'INT ENABLED'
00280          JP      114            ;RETURN TO BASIC 'READY'
00290 COUNT    LD      A,L            ;LOAD L REGISTER INTO A
00300          CP      255            ;SUBTRACT 255 & SET FLAGS
00310          JP      NZ,FINIS       ;IF NOT ZERO GOTO FINIS
00320          LD      A,(32102)      ;3RD COUNTER MEM LOCATION
00330          INC     A              ;ADD +1 TO A REGISTER
00340          LD      (32102),A      ;STASH IT AWAY IN MEM
00350          JP      FINIS          ;GOTO FINIS (= THE END)
00360          END     32200          ;INITIALIZE AT 32200
```

Figure 4-6

```
INTERRUPT DEMONSTRATION PROGRAM 2  -  OBJECT CODE

7D00             00120            ORG     32000
7D00  08         00130            EX      AF,AF'
7D01  D9         00140            EXX
7D02  2A647D     00150            LD      HL,(32100)
7D05  23         00160            INC     HL
7D06  22647D     00170            LD      (32100),HL
7D09  7C         00180            LD      A,H
7D0A  FEFF       00190            CP      255
7D0C  CACE7D     00200            JP      Z,COUNT
7D0F  08         00210 FINIS      EX      AF,AF'
7D10  D9         00220            EXX
7D11  FB         00230            EI
7D12  C9         00240            RET
7DC8             00250            ORG     32200
7DC8  ED56       00260            IM      1
7DCA  FB         00270            EI
7DCB  C37200     00280            JP      114
7DCE  7D         00290 COUNT      LD      A,L
7DCF  FEFF       00300            CP      255
7DD1  C20F7D     00310            JP      NZ,FINIS
7DD4  3A667D     00320            LD      A,(32102)
7DD7  3C         00330            INC     A
7DD8  32667D     00340            LD      (32102),A
7DDB  C30F7D     00350            JP      FINIS
7DC8             00360            END     32200
00000 TOTAL ERRORS
```

Figure 4-7

```
10 'BASIC PART OF HYBRID INTERRUPT DEMONSTRATION PGM 2
15 '
20 DEFINTY:CLS:POKE32100,0:POKE32101,0:POKE32102,0
25 PRINT@156,"INTERRUPTS  =";:PRINT@283,"BASIC COUNT  =";
30 POKE16402,195:POKE16403,0:POKE16404,125
40 PRINT@170,PEEK(32100)+256*PEEK(32101)+65536*PEEK(32102)
50 Y=Y+1:PRINT@298,Y:GOTO40
```

Figure 4-8

SOURCE CODE PROGRAM 2:

IS not all that different from the first one. Looking at the source code program in Figure 4-6, we see the EX AF, AF' and EXX instructions to save the registers in the BASIC program it comes FROM instead of the PUSH instruction. The instructions in lines 150, 160, and 170 perform the same functions as the first program's lines 140 to 160.

Lines 180-200: Check to see if the 'H' register has reached 255 and if so, GOTO COUNT in line 290.

Lines 290-310: Check to see if the 'L' register has reached 255 and if so, increments the 3rd storage location at 32102 by +1 in lines 320-340. Otherwise, the program loops back to FINIS.

Lines 210-230: Restore the orginal AF, BC, DE, & HL values before enabling interrupts and RETurning the interrupted BASIC program address off the top of the stack in line 240.

Lines 250-270: Are the same as our first interrupt program and allow us to set the Z-80 in interrupt mode one, enable interrupt flip-flop IFF1, and jump back to BASIC 'READY' in line 280.

By checking the value of the H register for 255, and THEN the value of the L register for 255, BEFORE incrementing MEM location 32102 by 1 = 65536 for each count, we have sneakily avoided using the ACCUM and CDBL Store in RAM......and having to save its contents when we jump back and forth from BASIC to the interrupt subroutine and then back to BASIC.

BASIC PART OF HYBRID PROGRAM 2:
Is shown in Figure 4-8 and also NOT all that different from the first one except for adding a bit of 'pretty printing labels' in line 25 and the additional multiplier provided by MEM location 32102. Running the program is also the same as first one. You will notice the interrupt subroutine is not quite as FAST as the first one due to the extra counter. Now let's take a giant leap forward (or backwards depending upon your viewpoint) and try a program that uses the ACCUM, video, and most ALL registers which must be saved and restored.

```
00100 ; VIDEO DISPLAY OF INTERRUPT
00110 ; SAVE EVERYTHING INCLUDING THE KITCHEN SINK:
00120 ; AF, BC, DE, HL, & IX REGISTERS
00130 ; ACCUM AND CDBL STORE IN TACUM 1 & 2
00140 ; CURSOR , LINE POSTION, AND CURSOR CHARACTER IN MEM
00150 ; RELOCATE STACK POINTER TO 31998 IN MEMORY
00160 START   EQU     7D00H             ;= 32000 DECIMAL
00170         ORG     START             ;PGM. BEGINS AT 32000
00180         EX      AF,AF'            ;SWAP ALT. AF REGISTERS
00190         EXX                       ;SWAP BC-DE-HL REGISTERS
00200         PUSH    IX                ;SAVE IX REG. IN STACK
00210         LD      DE,411DH          ;ACCUM ADDRESS
00220         LD      HL,TACUM1         ;TEMPORARY ACCUM STASH
00230         LD      B,8               ;BYTES TO MOVE
00240         CALL    09D7H             ;MOVE IT SUBROUTINE
00250         LD      DE,4127H          ;CDBL STORE ADDRESS
00260         LD      HL,TACUM2         ;TEMPORARY CDBL STASH
00270         LD      B,8               ;BYTES TO MOVE
00280         CALL    09D7H             ;MOVE IT SUBROUTINE
00290         LD      A,(40AFH)         ;NUMBER TYPE FLAG
00300         LD      (TNTF),A          ;SAVE NUMBER TYPE FLAG
00310         LD      HL,(4020H)        ;CURSOR POSITION IN MEM
00320         LD      (32400),HL        ;SAVE IT AT 32400/32401
00330         LD      A,(40A6H)         ;CURSOR LINE POSITION
00340         LD      (32402),A         ;SAVE IT AT 32402
00350         LD      A,(4022H)         ;CURSOR CHARACTER
00360         LD      (32403),A         ;SAVE IT AT 32403
00370         LD      HL,15369          ;VIDEO MEM LOCATION
00380         LD      (4020H),HL        ;MOVE CURSOR TO 15370
00390 VIDEO   DEFM    'INTERRUPT  CYCLES ='
00400         DEFB    0                 ;DEF MESSAGE DELIMITER
00410         LD      HL,VIDEO          ;MESSAGE LOCATION IN MEM
00420         CALL    28A7H             ;DISPLAY MESSAGE AT HL
00430         LD      A,2               ; 2 = INTEGER NUMBER TYPE
00440         LD      (40AFH),A         ;NUMBER TYPE MEM LOCATION
00450         LD      DE,(32404)        ;INTERRUPT COUNTER STORE
00460         INC     DE                ;ADD + 1 TO DE
00470         LD      (32404),DE        ;STASH IT AWAY IN MEM
00480         DEC     DE                ;SUBTRACT - 1 FROM DE
00490         LD      HL,1              ;+1 TO HL REGISTER
00500         CALL    0BD2H             ;ADD DE + HL IN ACCUM
00510         CALL    0FBDH             ;CONVERT ACCUM TO STRING
00520         CALL 28A7H                ;DISPLAY STRING ON VIDEO
00530 WAIT    LD      A,(14400)         ;MEM-ARROW KEYBOARD ROW
00540         CP      64                ;RIGHT ARROW PRESSED = 64
00550         JR      NZ,WAIT           ;GOTO WAIT TILL RT. ARROW
00560         LD      HL,(32400)        ;OLD CURSOR POSTION
00570         LD      (4020H),HL        ;RESTORE ORIGINAL CURSOR
00580         LD      A,(32402)         ;ORIG. LINE POSITION
00590         LD      (40A6H),A         ;RESTORE IT
00600         LD      A,(32403)         ;ORIG. CURSOR CHARACTER
00610         LD      (4022H),A         ;RESTORE IT
00620         LD      DE,TACUM1         ;TEMP. ACCUM. ADDRESS
00630         LD      HL,411DH          ;NORMAL ACCUM ADDRESS
00640         LD      B,8               ;BYTES TO MOVE
00650         CALL    09D7H             ;MOVE TEMP ACCUM BACK
00660         LD      DE,TACUM2         ;TEMP. CDBL STORE ADDRESS
00670         LD      HL,4127H          ;NORMAL CDBL STORE ADDR.
00680         LD      B,8               ;BYTES TO MOVE
00690         CALL    09D7H             ;MOVE CDBL STORE BACK
00700         LD      A,(TNTF)          ;TEMP NUMBER TYPE FLAG
00710         LD      (40AFH),A         ;RESTORE NUMBER TYPE FLAG
00720         EX      AF,AF'            ;RESTORE ORIGINAL REGS.
00730         EXX                       ; "       "        "
00740         POP     IX                ;RESTORE IX REG. FM STACK
00750         EI                        ;ENABLE INTERRUPT IFF1
00760         RET                       ;BASIC ADDRESS FM STACK
00770 TACUM1  DEFS    8                 ;8 BYTES ACCUM STASH
00780 TACUM2  DEFS    8                 ;8 BYTES CDBL STORE STASH
00790 TNTF    DEFS    1                 ;NUMBER TYPE FLAG STASH
00800         ORG     32200             ;CONT. PGM. AT 32200 DEC.
00810         LD      SP,START-2        ;MOVE STACK POINTER
00820         IM      1                 ;SET INTERRUPT MODE 1
00830         EI                        ;ENABLE INTERRUPT IFF1
00840         JP      114               ;RETURN TO BASIC 'READY'
00850         END     32200             ;INITIALIZE AT 32200
```

Figure 4-9

```
4 'BASIC PART OF HYBRID INTERRUPT DEMO PGM 3
6 '
10 DEFINTX
20 CLS:PRINT@137,"BASIC LOOP CYCLES ="
30 POKE16402,195:POKE16403,0:POKE16404,125
40 FORX=-32768TO32767
50 PRINT@156,X
60 NEXT
```

Figure 4-10

HYBRID DEMONSTRATION PROGRAMS - VERSION 3:

Source code is illustrated in Figure 4-9 and the BASIC segment in Figure 4-10.

We may have been a 'belt and suspenders' conservative in Figure 4-9's source code program by SAVING everything but the kitchen sink in the alternate registers pairs, the stack, and high MEM, plus moving the stack to 31998 in MEM. Obviously, this writer is a bit 'gun shy' from experience as not ALL of the variables and MEM locations had to be saved in this particular demo program. IF you have an interrupt and BASIC program that goes bananas when switching back and forth, you might check to see that you have saved everything conceivable before climbing up the wall or kicking your dog and cat.

We saved everything we could think of, and then restored it before returning to the BASIC program.....nevertheless, we probably overlooked something beween 14302 and 17129 decimal, though this program works perfectly.

The BRUTE FORCE APPROACH, which we stopped just short of using in this source code program would have been to move EVERYTHING to high MEM with the LDIR instruction, do our interrupt thing, and then move it back again before returning control to BASIC.

The comments with the source code program in Figure 4-9 are largely self explanatory. It operates much the same as the previous two demo programs EXCEPT it uses the 'RIGHT ARROW' key to allow you to single step through each interrupt if you wish.

By holding the 'RIGHT ARROW' key down continuously, the program will jump back and forth between basic and the interrupt subroutine depending on whether you are holding the iterrupt switch closed or not. In this 3rd version of the interupt exercise, the video display continually reads out the values of the interrupt counter and the BASIC program counter.

IF you have any difficulty following the source code program with its multiple CALLs to Level II ROM, by all means go back to Volume 1 of the 'Disassembled Handbook For TRS-80' and review the material, as it is ALL covered there with numerous demonstration programs to illustrate the CALLs used.

Hey there professor, why can't we use just the plain 'ole ldir instruction like you mentioned on the last page ? ? ?

My goodness Gridley, you are speaking 'Fagginsylvanian' again. Why not, not, indeed. Let's have a go at it

```
00100 ; MOVE EVERTHING INCLUDING THE KITCHEN SINK TO HIGH MEM
00110 ; USING THE LDIR INSTRUCTION AND THEN DO OUR INTERRUPT
00120 ; THING. THEN RESTORE EVERYTHING INCLUDING KITCHEN SINK
00130 ;
00140          ORG     32000          ;PGM. BEGINS AT 32000
00150          EX      AF,AF'         ;SWAP ALT. AF REGISTERS
00160          EXX                    ;SWAP BC-DE-HL RDGISTERS
00170          PUSH    IX             ;SAVE IX REG. IN STACK
00180          PUSH    IY             ;SAVE IY REG. IN STACK
00190          LD      HL,16384       ;BEGINNING MEM ADDRESS
00200          LD      DE,31000       ;PLACE TO MOVE IT
00210          LD      BC,744         ;TOTAL BYTES TO MOVE
00220          LDIR                   ;MOVE THEM
00230          LD      HL,15369       ;VIDEO MEM LOCATION
00240          LD      (4020H),HL     ;MOVE CURSOR TO 15370
00250 VIDEO    DEFM    'INTERRUPT  CYCLES ='
00260          DEFB    0              ;DEF MESSAGE DELIMITER
00270          LD      HL,VIDEO       ;MESSAGE LOCATION IN MEM
00280          CALL    28A7H          ;DISPLAY MESSAGE AT HL
00290          LD      A,2            ; 2 = INTEGER NUMBER TYPE
00300          LD      (40AFH),A      ;NUMBER TYPE MEM LOCATION
00310          LD      DE,(32404)     ;INTERRUPT COUNTER STORE
00320          INC     DE             ;ADD + 1 TO DE
00330          LD      (32404),DE     ;STASH IT AWAY IN MEM
00340          DEC     DE             ;SUBTRACT - 1 FROM DE
00350          LD      HL,1           ;+1 TO HL REGISTER
00360          CALL    0BD2H          ;ADD DE + HL IN ACCUM
00370          CALL    0FBDH          ;CONVERT ACCUM TO STRING
00380          CALL    28A7H          ;DISPLAY STRING ON VIDEO
00390 WAIT     LD      A,(14400)      ;MEM-ARROW KEYBOARD ROW
00400          CP      64             ;RIGHT ARROW PRESSED = 64
00410          JR      NZ,WAIT        ;GOTO WAIT TILL RT. ARROW
00420          LD      HL,31000       ;MEMORY STORE ADDRESS
00430          LD      DE,16384       ;RESTORE TO NORMAL ADDR
00440          LD      BC,744         ;BYTES TO MOVE BACK
00450          LDIR                   ;MOVE THEM ONE AND ALL
00460          EX      AF,AF'         ;RESTORE ORIGINAL REGS.
00470          EXX                    ; "      "       "
00480          POP     IY             ;RESTORE IY REG. FM STACK
00490          POP     IX             ;RESTORE IX REG. FM STACK
00500          EI                     ;ENABLE INTERRUPT IFF1
00510          RET                    ;BASIC ADDRESS FM STACK
00520          ORG     32200          ;CONT. PGM. AT 32200 DEC.
00530          IM      1              ;SET INTERRUPT MODE 1
00540          EI                     ;ENABLE INTERRUPT IFF1
00550          JP      114            ;RETURN TO BASIC 'READY'
00560          END     32200          ;INITIALIZE AT 32200
```

Figure 4-11

```
7D00            00140           ORG     32000
7D00 08         00150           EX      AF,AF'
7D01 D9         00160           EXX
7D02 DDE5       00170           PUSH    IX
7D04 FDE5       00180           PUSH    IY
7D06 210040     00190           LD      HL,16384
7D09 111879     00200           LD      DE,31000
7D0C 01E802     00210           LD      BC,744
7D0F EDB0       00220           LDIR
7D11 21093C     00230           LD      HL,15369
7D14 222040     00240           LD      (4020H),HL
7D17 49         00250 VIDEO     DEFM    'INTERRUPT  CYCLES ='
7D2A 00         00260           DEFB    0
7D2B 21177D     00270           LD      HL,VIDEO
7D2E CDA728     00280           CALL    28A7H
7D31 3E02       00290           LD      A,2
7D33 32AF40     00300           LD      (40AFH),A
7D36 ED5B947E   00310           LD      DE,(32404)
7D3A 13         00320           INC     DE
7D3B ED53947E   00330           LD      (32404),DE
7D3F 1B         00340           DEC     DE
7D40 210100     00350           LD      HL,1
7D43 CDD20B     00360           CALL    0BD2H
7D46 CDBD0F     00370           CALL    0FBDH
7D49 CDA728     00380           CALL    28A7H
7D4C 3A4038     00390 WAIT      LD      A,(14400)
7D4F FE40       00400           CP      64
7D51 20F9       00410           JR      NZ,WAIT
7D53 211879     00420           LD      HL,31000
7D56 110040     00430           LD      DE,16384
7D59 01E802     00440           LD      BC,744
7D5C EDB0       00450           LDIR
7D5E 08         00460           EX      AF,AF'
7D5F D9         00470           EXX
7D60 FDE1       00480           POP     IY
7D62 DDE1       00490           POP     IX
7D64 FB         00500           EI
7D65 C9         00510           RET
7DC8            00520           ORG     32200
7DC8 ED56       00530           IM      1
7DCA FB         00540           EI
7DCB C37200     00550           JP      114
7DC8            00560           END     32200
00000 TOTAL ERRORS


VIDEO   7D17 00250    00270
WAIT    7D4C 00390    00410
```

Figure 4-12

We saved almost 30 lines of source/object code in Figure 4-11's source code and Figure 4-12's object code compared to the program in Figure 4-9, but we sure paid a heavy price for it. What was the price we paid, Gridley?

Sure beats the heck out me. The program was twice as easy to write, used only about half as much memory, and used the same basic hybrid program in Figure 4-10 to run. Looks like we won the game to me.

Thank you for your clear and lucid thoughts, Gridley. The price you paid was the most precious one of all.....TIME.

The Z-80 LDIR instruction is mighty convenient to use when you want to move a considerable amount of memory around as we did in this program. Sad to say, it takes TIME to do it, and IF your interrupt subroutine must of necessity be faster than the average bear, then by all means use the longer program as it is well over 100 times faster. In most applications, the interrupt function is not being used for SPOOLING. That is, only doing its thing while some other SLOW accessory like a line printer (illustrated in the last Chapter), is printing a character or a paper-tape-punch recording a value for posterity.

SUMMARY:

We have only taken a real fast trip through Z-80 interrupt territory and just barely glimpsed SOME of the scenery that the Z-80's interrupt capability offers us. The constraints we laid out initially, NO CUTTING OR HACKING UP of the expansion interface, have been met....but they limited our tour to only the most FUNDAMENTAL applications and demonstration programs. That is why this Chapter was entitled an "Introduction To Interrupts." You have met the enemy, and WON, though the battle was a short one indeed.....really a skirmish, at best.

Wish to dig deeper? Order the SIO and PIO data books from Zilog. Read Barden's "The Z-80 Microcomputer Handbook," pages 29-39 and 104-115. They are singularly well written except for the comment on page 111, line 22, which is wrong. An external interrupt in IM 1 DOES NOT HAVE TO RECOGNIZE THE INTERRUPT ACKNOWLEDGE SIGNAL. It is only a matter of choice.

Write a BASIC program that provides its own interrupt table to 5 different interrupt subroutines located at 32100, 32200, 32300, 32400, and 32500. How to do it? Simply POKE the appropriate jump address into MEM locations 16403 and 16404 for each part of the BASIC program that would use a particular interrupt subroutine IF an interrupt arrived while that particular part of the BASIC program were executing.

Now, turn the page and have a 'go' at the questions for this Chapter. It was IMPOSSIBLE to write a difficult question on such a striaghtforward and interesting subject. We hope you have had as much pleasure reading this Chapter as we had writing it. Altogether, over 26 different programs were written to illustrate the points covered in this Chapter. Our volunteer students selected the four presented here as the BEST of the group to illustrate the points covered.

- CHAPTER 5 -

AN INTRODUCTION TO INTERFACING TO THE OUTSIDE REAL WORLD

INTRODUCTION:

Here we go again. Another introduction to an introduction. We do not wish to tackle the subject too abruptly for Gridley's sake, so we will just sort of ease into interfaces so slowly and gradually he will never know how far he has traveled until we are there. WAKE UP, Gridley. We are going to take a short journey into TRS80ville and Fagginsylavania.

Our tour will first start at PORT 255, where we will use the TRS-80's cassette port to interface with the outside world. We can use this port to both INput and OUTput control signals IF we follow a few simple rules and formats the TRS-80 is designed to utilize. The complexity of the external circuitry required can vary from only a simple Radio Shack relay and 1 or 2 TTL (transistor-transistor-logic) chip layout, on up to as complex as you wish to make it.

From PORT 255 we will go downhill all the way to PORT ZERO, where we will dissect the Telesis Labs model VAR/80 accessory in considerable detail and RUN a number of demonstration programs using this cost effective device. For those who are real adept soldering iron jockeys, we'll provide enough detail so you may 'roll your own' if you wish, on your own workbench.

In this Chapter, we will stick to plain 'ole digital ON or OFF signals and circuits (except for INput to PORT 255), and save analog to digital conversion techniques for the next Chapter. Any journey has to begin with the FIRST step. Come on Gridley, put one foot in front of the other. We're off.

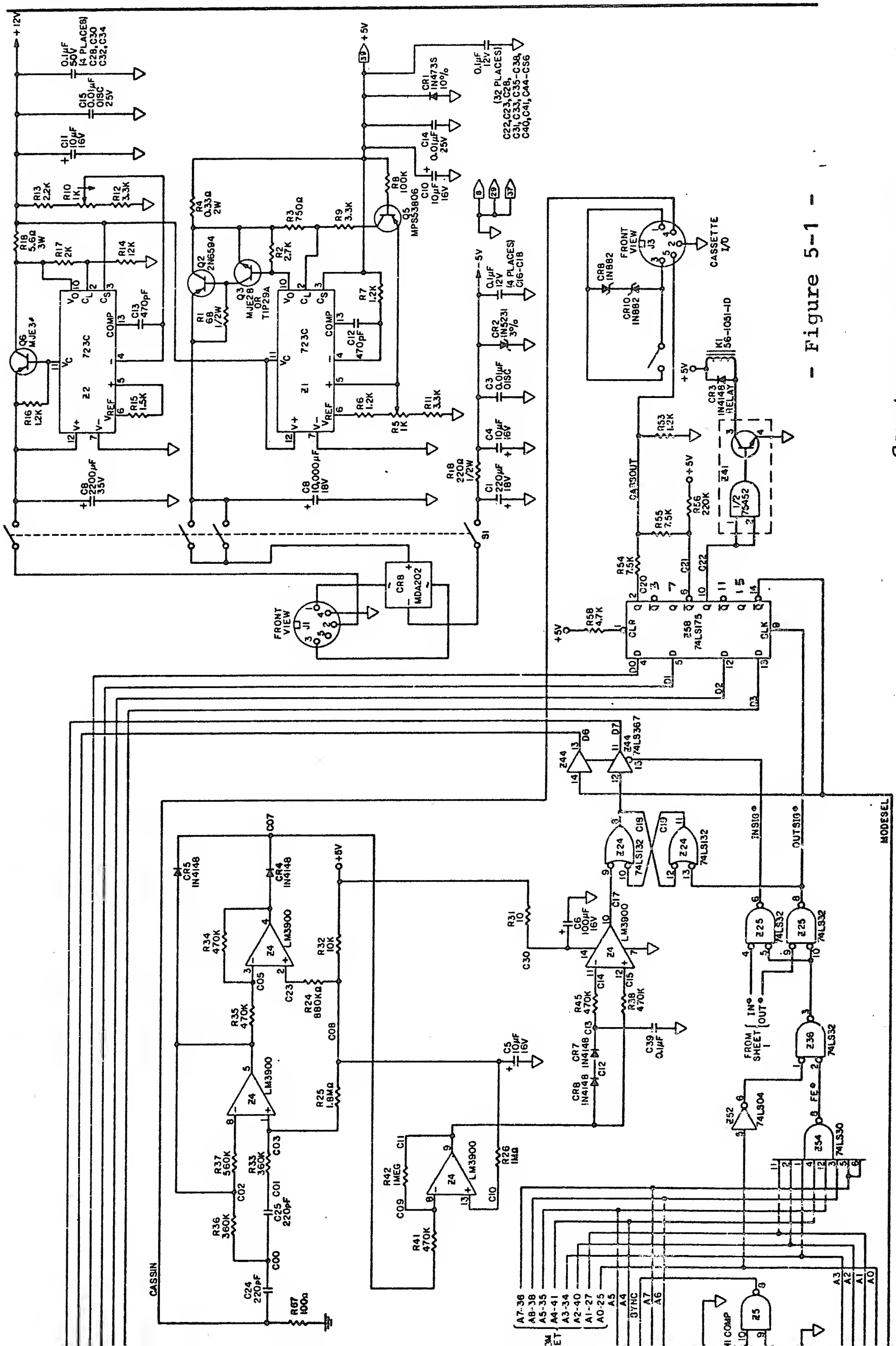
USING PORT 255 TO INTERFACE TO THE OUTSIDE WORLD:

Is surely the easiest place to visit to begin this Chapter. Steve Leininger and the other TRS-80 designers very kindly provided us with a 'Quad D Type Flip-Flop With Clear' at PORT 255 which normally controls our cassette input/output port, as well as the CHR$(23) BIG CHARACTERS command that puts our video display into the 32 characters per line mode. Heh, heh, heh....do you get it Gridley? (23) backwards is 32 characters per line? Those Ft. Worth cowboys have a real sense of humor.

What's so funny about that ? ? ?

It makes no nevermind, Gridley. I'll explain it to you later.

Let's take a look at the schematic for this section of our TRS-80 in Figure 5-1 on the following page. The upper-right hand quarter of the schematic is our keyboard's regulated power supply which we will ignore for the time being.



- Figure 5-1 -

Courtesy of Radio Shack

OUTPUT FROM PORT 255 AND 'HOW' IT GETS THERE:

Figure 5-1 illustrates Z59, a 74LS175 quad 'D' type flip-flop, operating as a combination data latch and SORT-OF modulator for PORT 255 OUTput to the cassette, as well as a data latch for the 64 or 32 characters per line MODESEL function. This little 49 cent Low power Schottky TTL integrated circuit is a very busy chip and is controlled by DATA lines 0, 1, 2, & 3 to select its mode of operation and the OUTSIG line which essentially turns it ON or OFF as its CLeaR pin is constantly held high (+5 Vdc) through resistor R59. It is an ingenious and extremely clever design by Radio Shack's Steve Leininger, formerly with National Semiconductor and of SC/MP micro computer design fame. Here is how it works.

On the lower left side of Fig. 5-1 we see Z52 and Z54 which are the PORT decoders using the lower 8 ADDRESS lines, A0 to A7. Whenever A0 to A7 are all 1's = 11111111 = 255 decimal AND the Z-80 derived OUT* line goes LOW (meaning out PORT 255), Z59 is turned ON by the OUTSIG line from flip-flop Z25, and does its thing. Its THING is determined by whatever DATA lines 0 to 3 instruct it to do. DATA lines 4 to 7 are ignored and make no nevermind to the Z59 chip. Let's see exactly what the signals on D0 to D3 accomplish whenever we have an OUT 255,xx instruction in BASIC or OUT (255),A in assembly language with 'A' register containing the value xx. We will use a few single line BASIC programs and a VOM, volt-ohmmeter, to measure what is going on here via the plugs to the cassette's AUX and REMOTE jacks. AUX = audio input to the cassette and REMOTE = cassette motor ON or OFF. Our video display will tell us whether we are in the 32 or 64 character per line mode. Since the output of each of the 4 flip-flops in Z59 is controlled solely by the value input by its respective DATA line, (assuming OUTSIG* has activated the Z59), and since there are NO further de-multiplexers in this circuit, we can have only 2 output states at each of Z59's 'Q' output pins. The NOT 'Q' (bar over 'Q') outputs for each flip-flop are simply the inverse of the plain 'Q' outputs and are sometimes called, 'double rail' output.

Hookup your VOM in the low range 'dc volts' mode to the plug going to your cassette's AUX jack, and try running the following one liner BASIC program:

```
10 FORX=1TO500:OUT255,1:NEXT:FORX=1TO500:OUT255,2:NEXT:GOTO10
```

You will notice that the output voltage swings back and forth between approximately .7 volts dc to about zero volts dc depending on whichever value the above program is placing on the DATA bus.......either a 1 or a 2. This is a sort-of very 'slow motion' version of the burst modulated audio signals that the CSAVE function records on cassette. The 1 milli-second 'burst' periods AND the amplitude of the audio signal is controlled by the Level II ROM program. It is an ingenious way to store digital data with inexpensive cassette recorders.

Now switch the volt-ohmmeter leads to the cassette REMOTE cord's mini-jack and set the VOM to read low-scale ohms. Load and run the following BASIC one liner:

```
10 FORX=1TO500:OUT255,4:NEXT:FORX=1TO500:OUT255,0:NEXT:GOTO10
```

What happened, Gridley? Please wake him up.

Well, almost the same thing happened. Except this time, the cassette reed relay K1, is closing during the OUT255,4 part of the program and opening during the OUT255,0 part.

Let's go one step further and run this mini-program. Leave the program on the video display WHILE you run it.

```
10 FORX=1TO500:OUT255,8:NEXT:FORX=1TO500:OUT255,0:NEXT:GOTO10
```

WHAT HAPPENED GRIDLEY?

I'm awake. I'm awake. Need not speak so LOUDLY. The crazy video display went bananas is what happened. It jumped from 32 characters per line BIG display to 64 characters per line normal display, and then back again, is what occurred.

Verrrry good, Gridley. Now let's see what conclusions we can draw from the last few little one liner experiments about using PORT 255 after taking a look at the following picture.

DECIMAL -	DATA BUS CONTENTS			
NUMBER	D3	D2	D1	D0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0

When using the OUT 255 instruction, the two least significant bits on the DATA bus, D0 and D1 control the BOTH the switching period (frequency) AND the modulation level of the signal going to our cassette recorder. The third least significant bit, D2 controls the closed-open status of K1, the reed relay that turns the cassette recorder ON and OFF. Lastly, the fourth least significant bit D3, controls our characters per line selection of 32 or 64 via the MODESEL output on the bottom of Figure 5-1. AGAIN, the values on the DATA bus lines D4, D5, D6, and D7 are ignored since Z-59 is only connected to D0, D1, D2, and D3. Give you any ideas, Gridley?

You bet it does. What if we hooked up another 49 cent 74LS175 chip, just like Z59, to the DATA bus lines D4, D5, D6, & D7, plus this new chip's "clock" and "clear" pins to the ones on Z59? Wouldn't it work? Wouldn't it give us 4 more outputs?

Gridley, you never cease to amaze me. What an outstandingly creative and inventive creature you can be at times. Of course your idea will work. It would give us 4 more independently controllable outputs from PORT 255 that could be activated with:

DECIMAL NUMBER	- DATA BUS OUTPUT D7	D6	D5	D4	D3	D2	D1	D0
16	0	0	0	1	0	0	0	0
32	0	0	1	0	0	0	0	0
64	0	1	0	0	0	0	0	0
128	1	0	0	0	0	0	0	0

By running each of the 74LS175's 'Q' outputs to a single 74LS154 4-line to 16-line demultiplexer, we could obtain 16 separate independently controllable outputs from PORT 255 and in NO way interfere or bolix up those coming from D0, D1, D2, & D3. The following chart illustrates the DMUX, demultiplexer outputs we could obtain, versus the OUT 255,value we would use in decimal to activate the output channel desired:

DECIMAL NUMBER	- DATA BUS OUTPUT D7	D6	D5	D4	D3	D2	D1	D0	- DMUX CHANNEL
<16	0	0	0	0	0	0	0	0	0
16	0	0	0	1	0	0	0	0	1
32	0	0	1	0	0	0	0	0	2
48	0	0	1	1	0	0	0	0	3
64	0	1	0	0	0	0	0	0	4
80	0	1	0	1	0	0	0	0	5
96	0	1	1	0	0	0	0	0	6
112	0	1	1	1	0	0	0	0	7
128	1	0	0	0	0	0	0	0	8
144	1	0	0	1	0	0	0	0	9
160	1	0	1	0	0	0	0	0	10
176	1	0	1	1	0	0	0	0	11
192	1	1	0	0	0	0	0	0	12
208	1	1	0	1	0	0	0	0	13
224	1	1	1	0	0	0	0	0	14
240	1	1	1	1	0	0	0	0	15

Actually, the values of D0, D1, D2, & D3 are "don't care" values to this particular circuit, and are included as zeros only for the sake of clarity since our demultiplexer inputs are only connected to the outputs of the additional 74LS175 which is connected to DATA bus lines D4, D5, D6, & D7. This circuit may be built for about $3 in a few hours and requires only one caution; i.e., the keyboard's power supply is already working at near MAXIMUM capacity, so use a SEPARATE +5 volts dc supply to power these 2 chips and any additional gates and/ or relays you may be adding. For controlling the maximum number of external devices with the TRS-80 at MINIMUM cost, this has got be the most cost-effective solution.

BACK TO FUNDAMENTALS AGAIN:

Let's assume that you DO NOT wish to build the foregoing gizmo

NOR have need for more than a single output to the outside real world. There are many avenues open to us varying in complexity from just adding a single Radio Shack $2.99 buffer relay and battery to power it that is activated by the cassette ON-OFF relay K1, from the cassette remote plug, on up to opening the 32/64 character per line MODESEL trace from Z59 and using it control the external device, IF you have no use or need for the large 32 character per line display mode. Here are the ways and means to implement some of the options, plus their inherent advantages and disadvantages.

1. REMOTE BUFFER RELAY ACTIVATED BY RELAY K1:
This is the ultimate in simplicity and requires only a Radio Shack #275-004 6 volt dc relay, a 9 volt battery, and a 390 ohm 1/4 watt dropping resistor. Even though the contacts of K1 are protected from voltage spikes (and damage) by Zener diodes CR9 and CR10, we added a single 1N4148 across our new buffer relay's coil for good measure. Relay K1, a subminiature reed type relay (on the bottom left side of the keyboard printed circuit board-round yellow tubular package) is rated to carry up to 500 milliamps at about 5 volts dc. Even so, it is one of the most failure-prone components in the entire assembly. In this particular application it is only carrying about 8 milliamps to power the new buffer relay, so should suffer no harm whatsoever. The new relay's contacts are rated at 125 volts at 1 amp. If you need to control a higher voltage OR current, by all means add another buffer relay with adequate contact ratings. Figure 5-2 illustrates the hookup.

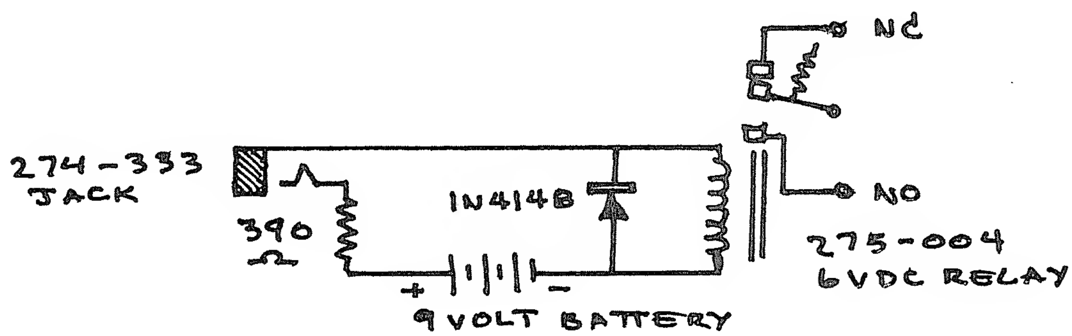


Figure 5-2

The battery is an ordinary 9 volt dc transistor radio battery that should handle the 8 milliamps current drain adequately for intermittent operation. If you are careful, the relay may be scotch taped directly to the battery, with the Radio Shack 274-333 mini-phone jack's leads also scotch taped to the 9 volt battery. To operate, just unplug the mini-phone plug from your cassette recorder and plug it into the buffer relay-battery assembly and you are ready to 'go.' The relay and circuit are NOT 'latching' in this configuration; i.e., will not stay closed unless put inside a FOR-NEXT or JR,NZ loop. To 'latch' simply add a Radio Shack #275-214 12 volt dc relay with one set of the contacts feeding 12 vdc to its coil when closed and a normally closed Radio Shack #275-148 pushbutton switch to reset. The time constant of the actuating loop must only be long enough to allow the contacts to close.

AGAIN, to activate this circuit we would either put OUT255,4 in the middle of a FOR-NEXT loop for as long as we wished it turned ON, or in the middle of an assembly language subroutine loop with OUT(255),A ('A' register should = 4).

ADVANTAGES OF THIS CIRCUIT:
It may be built for only a few dollars in a few minutes. It does NOT require you to open the keyboard case. A 12 year old Boy Scout has built and operated it successfully for keying a Novice Class ham radio transmitter using Volume 2's Morse Code program in Chapter 10.

DISADVANTAGES OF THIS CIRCUIT:
You must physically unplug the cassette's REMOTE plug and plug it into Figure 5-2's circuit. For high speed switching it is rather SLOW since TWO electromechanical relays have to be actuated. For Morse Code applications it is limited to about 15 words per minute code speed with the relays noted.

2. USING Z59'S NOT 'Q' (BAR) PIN #11 FOR RELAY CONTROL:

You will recall that Z59's output for each of the 4 flipflops is of the 'double-rail' variety; i.e., for NO input, one output is LOW and the other output is high. Pin #11, the NOT 'Q' (bar) output for the 3rd flipflop that controls cassette relay K1 is not used by the TRS-80. By running pin #11's signal to one of the six TTL inverters in a type 7406 buffer chip (high input = low output and vice versa), and then running this buffer's output to a high-speed keying relay, we can overcome APPROACH #1's disadvantages. The 7406 TTL chip has high-voltage outputs that can handle most any high-speed keying relay requiring 12 to 24 volts dc at up to 40 milli-amps per inverter. If you need more current to drive your keying relay (unlikely) simply parallel inverters within the chip. Driving this circuit is identical to illustration #1.

ADVANTAGES OF THIS CIRCUIT:
Speed. The Morse program in Volume 2 will TRANSMIT excellent Morse code up to about 35 to 40 words per minute which is far faster than most radio amateurs can copy.

DISADVANTAGES OF THIS CIRCUIT:
You must open the keyboard to hook it up. We use a Radio Shack mini-phone jack, #274-251, mounted on the rear apron of the keyboard to connect the 7406 chip and keying relay to Z59's pin #11 and ground. IF the cassette recorder is attached with its PLAY key depressed, the recorder will RUN whenever your Morse code or external relay control program is being activated. This is not a disadvantage. Only a reminder to turn it OFF.

3. USING ALL OF Z59'S OUTPUTS TO DRIVE INDIVIDUAL RELAYS
It is a relatively easy matter to bring all 4 of Z59' outputs to normally-closed mini-phone jacks on the rear apron of the TRS-80 keyboard. First the traces from Z59's pins 2, 6, 10,

and pin 14 are cut open with an X-Acto razor knife wherever you find a convenient spot that is OUTBOARD of the chip. Bring the Z59 side of each trace on a wire out to a normally-closed mini-phone jack (4 each) Radio Shack #274-253, that are mounted in 1/8" diameter holes drilled into the rear apron of the keyboard. Each of these 4 wires are soldered to the most clockwise pin of a mini-phone jack (looking at it from the rear with the pins, up). The otherside of each trace from Z59 that was cut open is then brought out on a wire to the center pin of each mini-phone jack and soldered. Also bring a wire from the printed circuit board GROUND to each jack and solder it to the most counter-clockwise pin. Mount the jacks to the keyboard.

What we have now is an output line from each of the four Z59 flipflops coming out to an individual normally closed jack on the rear of the TRS-80 keyboard, and then continuing THROUGH the jack and back to its normal location on the cut trace. With NO plugs in these jacks, nothing has been changed and Z59's outputs all operate as before. By attaching 7407 TTL buffers (high input = high output) via mini-phone plugs to those jacks connected to Z59's pins 2 and 10, and 7406 TTL buffer-inverters (low input = high output) via mini-phone plugs to those jacks connected to Z59's pins 6 and 14, we would now have 4 individual TTL high voltage open-collector outputs capable of handling most anything up to 30 volts dc and each capable of sinking up 40 milliamps. Each could be activated by an OUT 255,1 or OUT255,2 or OUT255,4 or OUT255,8 respectively.....or in combination if desired.

ADVANTAGES OF THIS CIRCUIT:
Four individually controlled outputs to the outside world. Very low cost; about $2 or $3 plus relays. Plug-in when desired....unplug when not being used.

DISADVANTAGES OF THIS CIRCUIT
Must open keyboard and cut traces. Cannot use cassette or 32 characters per line mode when being used. Must be plugged in to activate.

QUESTION ! ! ! Why not use unused pins 3 and 7 of the first 2 flip flops, pin 11 of the third flipflop, and forget about using the fourth flipflip to avoid messing up the 32 character per line mode ? ? ? With the cassette turned OFF, wouldn't that give us 3 indvidually controlled outputs to the outside world AND keep us from cutting any traces AND keep us from having to plug-in and unplug all those crazy wires and plugs when we wished to use it ? ? ?

ANSWER: YES, to all your questions, Gridley. You are back to your most brilliant BEST, again. This is really your THING and your BEST Chapter so far in Volume 3. In this case, we would use a 7406 inverter-buffer on pins 3 and 11, and a 7407 buffer on pin 7. It will work just fine with the cassette turned OFF and SHOULD not interfere, we believe, in any way with normal cassette operation. This is a splendid idea, Gridley.

ANOTHER QUESTION ! ! ! I should quit while I'm ahead, but WHY couldn't we use an ordinary non-high voltage buffer-inverter like a 74LS04 on the NOT 'Q' output pins 3 & 11, and hook them plus 'Q' output pin 7 ALL to a 3-line to 8-line demultiplexer like a 74LS138 ? ? ? Wouldn't that give us 8 eight independently controllable outputs to the outside world? Same as before, without having to cut traces, plug-in all sorts of wires, and all that foolishness?

ANSWER: YES, to all your questions, Gridley. The ONLY requirement we can see is that the cassette must be turned OFF while you are using this system, which is certainly no problem. The chart below illustrates the 74LS138 demultiplexer's 8-line output versus 3-line input. Use 7406/07 TTL buffer-inverters tied into the DMUX output to drive relays as required

INPUT			-	DMUX OUTPUT							
1	2	3		Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
L	L	L		L*	H	H	H	H	H	H	H
L	L	H		H	L*	H	H	H	H	H	H
L	H	L		H	H	L*	H	H	H	H	H
L	H	H		H	H	H	L*	H	H	H	H
H	L	L		H	H	H	H	L*	H	H	H
H	L	H		H	H	H	H	H	L*	H	H
H	H	L		H	H	H	H	H	H	L*	H
H	H	H		H	H	H	H	H	H	H	L*

* Are only to highlight DMUX OUTPUT for a given binary input.

ANYMORE OPTIONS ? ? ? MY GOODNESS, WE'VE GOT 24 SEPARATE AND INDEPENDENTLY CONTROLLABLE OUTPUTS FROM PORT 255 SO FAR USING THE 74LS154 COVERED A FEW PAGES BACK FROM WITH D4 - D7, PLUS THE ONE SHOWN ABOVE USING THE 74LS138 ON UNUSED PINS OF Z59 ?

Calm down, Gridley. You are certainly flushed with success. There is only ONE more worth mentioning.

4. USING TWO 74LS154s - ONE ON Z-59'S FOUR OUTPUTS AND ONE ON ANOTHER 74LS175 CONNECTED TO D4 - D7.

Well now, we have almost come full circle from the idea a few pages back of using a single 74LS154 driven by a 74LS175 on D4 through D7. Why not use another 74LS154 on option #3's four outputs from Z59? Why not, indeed. If for some weird reason or another you MUST have between 25 and 32 separately controlled outputs to the outside world, then this is probably the easiest solution. Parts cost is around $9 at the outside, PLUS (and a very expensive plus) the cost of whatever relays you wish to drive with the 7406/07 high-voltage open-collector buffer chips. The advantages and disadvantages of this type of hookup are pretty well covered by #3 option.

We could go on and on with demultiplexing all of PORT 255's possible 256 (including zero) combinations of outputs, but the 1 to 32 separately controllable outputs from this port that we have covered so far, should meet most readers' needs.

USING PORT 255 TO INPUT DATA FROM THE OUTSIDE REAL WORLD:

Is ALMOST as simple as using PORT 255 to output data IF you only need a SINGLE digital, yes or no, high or low, input.

Take a look at Figure 5-1 again. Here we see Z54 and Z52 decoding the lower 8 bits of ADDRESS bus A0 through A7 which carry the PORT address for either an OUT or IN(P) instruction in either BASIC or assembly language. Whenever we have a low at pin 3 of Z36 (= PORT 255 being addressed) AND a low at pin 4 of Z25 (= INP instruction), then Z25 will output a low to tri-state hex buffer Z44's control pin 15, thus turning Z44 ON; i.e., high in = high out and low in = low out of Z44. IF Z44's control pin is high, it presents a high impedance to DATA bus lines D6 & D7 who THEN do not know it is even there. Let's assume that the instruction is INP(255) and follow the signal in from the cassette connector jack, J3's pin 4 on the lower right corner of Figure 5-1.

This is the CASSIN line that squirrel's its way around to the upper left corner of Figure 5-1 and finally feeds into the first stage of Norton quad opamp Z4. This first stage is an active high pass filter that passes most signals between 500 and 2500 cycles, thus getting rid of 60 cycle hum and any high frequency (above 3000 cycles or so) noise that may have been picked up. The second opamp section of Z4 serves as an active detector with positive dc pulse output that drives the third opamp in Z4. This third section with input at pin 8 then both inverts (positive is now negative) and amplifies the signal to drive the fourth and last opamp in Z4 through diodes CR6 and CR7 that feed its pin 11 through a 470K ohm resistor, R5. This fourth and last opamp serves as a level detector and outputs a standard TTL level signal of nominally +4.4 volts dc or zero volts dc = signal NOT present or present) on its pin 10 to flipflop Z24. Z24 is acting as a 'latch' (just like a relay's latching action) with its pin 8 driving Z44 tri-state buffer which when ON feeds DATA bus line D7. Take a deep breath, Gridley.....we are almost there.

Now, there are TWO easy ways we can hookup the TRS-80 to SENSE whether or not an external device is putting out a signal or not putting out a signal to PORT 255. The first way does NOT require any cutting, hacking, or soldering of our precious keyboard. It only requires that we provide a 500 cycle to 2000 cycle audio tone of about 1 volt peak to peak, to the cassette EAR line plug that we would normally have plugged into the recorder. This audio tone can be generated by a single LM555 chip (Radio Shack #276-1723 @ 79 cents each), a Morse code practice oscillator, or even the audio from a ham radio receiver that we used in Volume 2's Chapter 10.

REMEMBER GRIDLEY, THIS DISCUSSION ON INPUTTING A VALUE VIA PORT 255 IS "NOT" FROM CASSETTE, BUT "ONLY" FROM AN EXTERNAL SOURCE THAT IS "EMULATING" (looks like), THAT FROM CASSETTE.

I do, I do, recognize that fact. No need to shout at me.

USING AN AUDIO SIGNAL TO RING PORT 255'S "INPUT" BELL:

Is just plain downtown Simplesville. All we need is about 1 volt peak to peak of audio signal ANYWHERE in the frequency range of 500 to 2000 cycles. IF you have the Radio Shack E-Z cassette load modification installed in your keyboard (2 chip modification 'stuck' onto the keyboard PCB about front and slightly right of center), you will only need about 250 milli-volts of audio, though it will work just as well with 1 volt too. Since Gridley is a theoretician and not too handy at the workbench, we will use the Radio Shack code practice oscillator, #20-005 to generate the audio signal. The only modification required is to add a mini-phone jack, #274-253 to the speaker terminals so that when the EAR line to the cassette recorder is plugged into it, the code practice oscillator's speaker terminals are OPENED and the full audio signal of about 1 volt peak to peak is fed into the EAR line. Drill a 1/8" diameter hole in the side of the case and mount the mini-phone jack on the code practice oscillator.

Now, with everything hooked up, try the following BASIC one liner. Then, RUN, and press the code oscillator's (or your own) key, DOWN. What happened, Gridley ? ? ?

```
10 PRINT@158,INP(255):GOTO10
```

NOTHING HAPPENED ! ! ! All the program output was a '127.'

Ok, Gridley. Try it again and keep turning UP the volume until you receive a CLUE. What happened, Gridley ? ? ?

SONOFAGUN, IT OUTPUT A 255, AND STAYED THERE. WHY ? ? ?

Well Gridley, Z24 is a LATCH. It 'locks' the signal into DATA bus line D7 via Z44 until Z24 flipflop is reset. Let's try to RESET Z24 by adding the following statement to our program and RUN. What happened, Gridley ? ? ?

```
10 PRINT@158,INP(255):IFINP(255)>200THENOUT255,0
20 GOTO10
```

IT WORKED ! EVERY TIME I PRESSED THE KEY DOWN IT OUTPUT '255' AND THEN RETURNED TO '127' WITH THE KEY UP.
WHY DID YOU USE ">200" RATHER THAN "=255" TO RESET Z24 ? ? ?

A good question, Gridley. Some TRS-80s (a few) have a strange habit of putting out most any number > 200 when a signal is input on the cassette line and '127' for no signal. This seems to solve the problem for these WEIRD ones and does not foul-up the normal ones.

The primary point to note is that the OUT255 does indeed RESET Z24 via Z25's pin 8 which drives Z24's pin 13, thus allowing the next INP(255) signal to 'toggle' it via pins 9 and 8 (toggle as in toggle switch = turn it 'ON' or turn it 'OFF.'

ANY PRACTICAL APPLICATIONS FOR WHAT WE HAVE LEARNED SO FAR ?

Sure there are IF you have need to INPUT the ON or OFF status, YES or NO status, or 1 or 0 status of some device in the outside real world. As far as 'practical' applications are concerned, they are left entirely to your own imagination or needs, if you will. Volume 2's Chapter 10 covered in detail a BASIC program that uses the Morse code audio signal from an amateur radio receiver's speaker terminals to print out the decoded Morse signal in alphanumerics on the TRS-80 video display. In the RECEIVE mode, the program would decode and print out the message of properly spaced Morse signals (dots and dashes) up to about 20 words per minute. Certainly not FAST, but typical for the average radio amateur who transmits in the 15 to 20 words per minute ball park. Though this Morse code program has never been extensively advertised in ham radio publications, over 1000 copies in both cassette and disk formats have been purchased by amateurs throughout the world. It is available from Richcraft on cassette or disk for $15. postpaid for those who do not wish the pleasure of entering two 8000 byte programs via their own keyboard. Its principal merits are that it WORKS and the PRICE IS RIGHT.

A SECOND APPROACH TO RINGING PORT 255'S BELL:

This approach is for those computerists who do not mind opening up and cutting a trace on their keyboard. It works quite similarly to our first approach, but does not require an audio signal to activate DATA bus line D7. All we do in this case is cut the trace between Z4's output pin 10 and Z24's input pin 9. Each side of the cut trace is brought out to a normally closed #274-253 mini-phone jack mounted on the rear apron of the keyboard. A mini-phone plug and line when inserted, bring out both printed circuit board ground and the input to pin 9 of Z24. A LOW (zero volts dc) on this line places a '1' on DATA bus line D7 via Z24 when 'set' and via Z44. A HIGH (4.4 to 5.0 volts dc) on this line places a '0' on DATA bus line D7 via Z24 when 'set' and via Z44.

Depending on your viewpoint, the merit of this second approach is that it DOES NOT require an audio oscillator to implement it, as did our first approach.

ANY OTHER QUICK AND CHEAP AND DIRTY APPROACHES TO INP(255) ? ?

There sure are, but since this Chapter still has a LONG way to go, we will only outline one for you; i.e., food for thought for the inveterate 'builder' of his/her own gizmos who wishes to 'dig deeper' into using PORT 255 for INPutting data. The easiest way is to hang-on a 74LS367's hex buffers outputs to DATA bus lines D0 to D5. Parallel Z25's INSIG* output to the new 74LS367's control pin 15. If you wish a 'latched' input, use a 74LS174 type D hex flipflop with clear for your incoming signals. Sure sounds easy enough & it is. Good luck, Gridley.

INTERFACING TO THE OUTSIDE WORLD WITH THE TELESIS VAR/80:

This is one of the best relatively low cost interfaces to the outside world for the TRS-80. It provides 8 'latching outputs' and 8 'non-latching' inputs nominally via PORT zero. We will cover what we mean by nominally later. Two of the outputs are via single-pole, double-throw, heavy duty relays that can handle 3 amps at 120 volts ac with normally-closed and normally-open contact terminals brought out to the front panel. The other 6 outputs deliver TTL logic levels of zero and +5 volts dc, also brought out to the front panel. Figure 5-3 is a photo of the front panel.

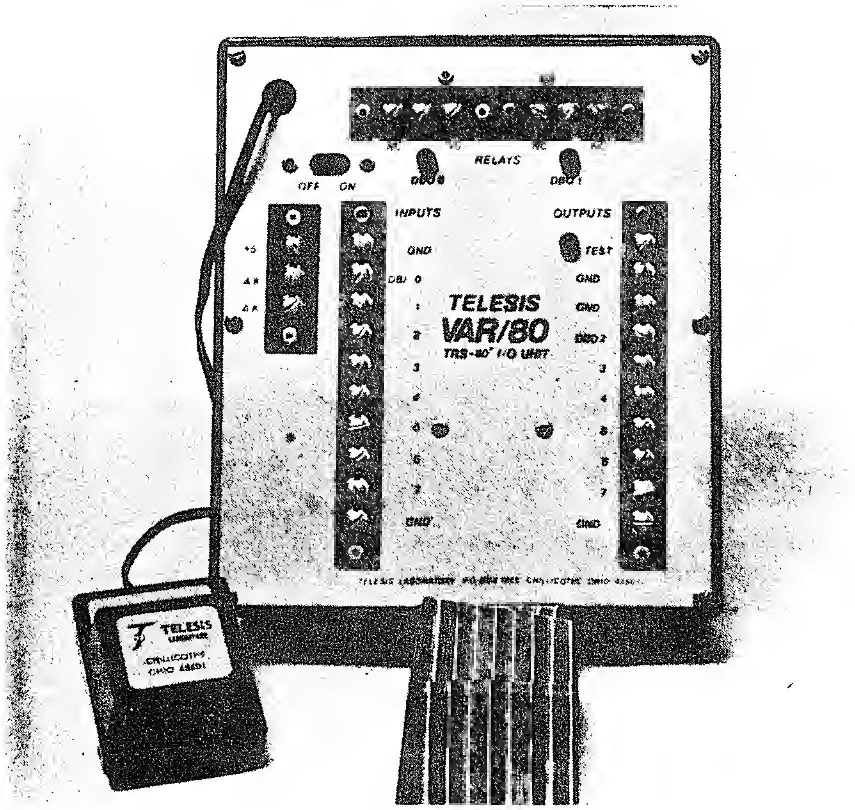


Figure 5-3

The upper horizontal terminal strip shown in Figure 5-3 contains the terminals for the two output relays. Just beneath this terminal strip are two 'red' light emitting diodes that come 'ON' whenever their respective relay is activated. The vertical right-hand terminal strip contains the terminals for the additional six latched TTL outputs, plus a TEST 'LED' and terminal. More about this later.

The longest vertical left-hand terminal strip contains the terminals for the eight INPUTS, and the short terminal, the 'anode' connections to the two 'opto-isolated' inputs and a +5 volt dc output for TESTing any of the inputs. The front panel and the system's box are BIG and easy to work with. The panel is 7" wide by 8" high, and the box measures 3" deep.

The 40 conductor 2 foot long cable and gold-plated female connector to the TRS-80 bus are brought out at the bottom of the system's front panel and can be seen in Figure 5-3. At the bottom left of Figure 5-3 is the 'plug-in' transformer power supply for the unit. It is switchable betweem 120 volts ac input @ 60 cycles or 220 volts ac input @ 50 cycles. The many European TRS-80 users will appreciate this feature.

HOW MUCH IT COSTS AND WHERE YOU CAN OBTAIN THE VAR/80:

As of September 1980, the VAR/80 sells for $ 109.95 + $3. shipping/handling and is available from:

Telesis Laboratory
100 R & D Drive - Box 1843
Chillicothe, Ohio 45601

Phone: (614) 773-1414

NO, THIS IS NOT A COMMERCIAL FOR TELESIS LABORATORY:

Why we selected the VAR/80 over its competitors for inclusion in Volume 3? Three significant reasons:

1. Reliability. We have used VAR/80 serial no. 0009 for nearly two years with no failures of any variety. It is extremely well built and designed very conservatively.

2. Ease of access...easy to use...uncluttered...uncrowded. So very many accessory designers these days try to crowd the maximum number of components into the smallest case/box possible. The result is an accessory that requires a magnifying glass to work on, and components working at higher temperatures = reduced reliability.

3. Extra TEST features for both INPUT and OUTPUT are built into the system with direct readout on the front panel. No volt-ohmmeters or haywire gizmos are required to TEST each and every channel.

WHAT IS INSIDE THE 'BIG' BLACK BOX ? ? ?

Lots of goodies, Gridley. Let's run through them briefly.

- full wave rectifier and 5 volt dc regulator chip for the plug-in-able power transformer with 10-12 volts ac output.
- (2) 74LS75 (X1 & X2) quad latches with double rail output
- (1) 74LS28 (X3) quad 2-input NOR buffer
- (2) 74LS367 (X4 & X5) tri-state hex buffers
- (2) GE 807-H11B3 optical-isolators with 200+ volt rating
- (2) 2N5089 transistors to drive the two 12 volt dc relays
- (2) Cornell-Dubilier 12 volt dc relays with 3 amp contacts
- (1) 2N4401 transistor
- assorted resistors, capacitors and BIG printed circuit board

Figure 5-4 is a drawing of the front panel that is easier to read than than Figure 5-3's photograph.

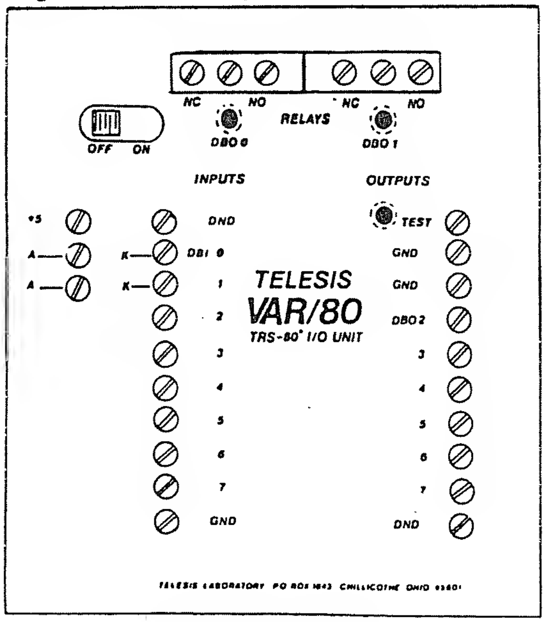


Figure 5-4

HOW DOES IT WORK ? ? ?

Pretty much the same way as PORT 255 described earlier, Gridley. BUT, and this is a big but, it only decodes address bus line A3 instead of all eight lower address lines A0 through A7. This was done by the designer, Victor A. Rizzardi (now we know what the 'VAR' stands for) at Telesis Labs for two reasons:

1. To minimize address line loading since when the VAR/80 was designed, the keyboard/expansion interface BUFFER modification had not yet been introduced.

2. To save the cost of an additional integrated circuit chip and allow ALL the extra features previously mentioned.

IS THAT A PARTICULAR DRAWBACK ? ? ?

No Gridley, I think not. Actually it is an advantage IF you are using the VAR/80 WITHOUT the expansion interface and its BUFFER/cable modification, since you can plug it directly into the keyboard's 40 pin output slot and NOT overload the address bus. When used with the expansion interface, the VAR/80 is plugged into the screen printer connector.

IF IT ONLY DECODES ONE ADDRESS LINE, HOW DO WE ADDRESS IT ? ?

Well Gridley, when all else fails, we can follow instructions and use OUT0,xxx or INP(0). Actually, since only one address line is decoded we may use ANY of the following PORT numbers: 0-7, 16-23, 32-39, 48-55, 74-71, 80-87, 96-103, 112-119, 128-135, 144-151, 160-167, 176-183, 192-199, 208-215, 224-231, AND 240-247. Notice that it DOES NOT interfere with PORT 255.

WHAT IF I NEEDED TO USE 'ALL 255' DIFFERENT PORTS ? ? ?

I'll answer that question with a question, Gridley. What have you got in mind? Controlling ALL the traffic lights in New York City?

NO, BUT 'WHAT IF.' ? ? ?

Ok, Gridley. Here is the clue. I won't draw any pictures since this Chapter is TOO long as it is. You go out and BUY a 74LS30 8-input NAND chip just like the TRS-80 uses for Z54. They only cost 59 cents from Radio Shack and YOU can easily wire it onto the VAR/80 printed circuit board with each input going to a low order address line, D0 to D7, EXCEPT add a 74LS04 inverter/buffer IN SERIES with each input line to the 74LS30 that you wish to decode as a LOW. Forinstance, if you wished to decode PORT 254 = 11111110 binary, then you would wire in a 74LS04 inverter/buffer gate from address bus line A0 to ANY of the 74LS30 input pins. Got it, Gridley ?

YES & NO. HOW DO I KNOW WHICH PIN ON THE VAR/80 END IS WHICH?

Very well, Gridley. ONE picture, but that is it. See below.

```
      38*  34*  30*  26*  22*  18*  14*  10*  6*  2*

     37*  33*  29*  25*  21*  17*  13*   9*  5*  1*

    40*  36*  32*  28*  24*  20*  16*  12*  8*  4*

   39*  35*  31*  27*  23*  19*  15*  11*  7*  3*
```

The above layout has the 40 pin BUS numbers FROM either the keyboard's rear connector OR the expansion interface screen printer connector. They are illustrated as viewed from the BOTTOM of the VAR/80 printed circuit board cable termination. A description of each pin's contents is given on page 228 of Dave Lien's (Radio Shack's) "User's Manual For Level 1." Hopefully, you can find it, Gridley?

! ! ! S-I-L-E-N-C-E ! ! !

Well now, let's press on. One comment on changing PORT decoding numbers with NO modification EXCEPT moving the connection FROM pin 34 above, which is address bus line A3. IF we moved this connection to pin 25 which is address bus line A0, THEN the VAR/80's single decoder would activate with PORT 0, 2, 4, 6, 8, 10, etc., but NOT 255 as we do not want our VAR/80 to ding-a-ling every time we use the cassette or 32 characters per line mode that ALL use PORT 255...an ODD number.

I CAN'T FIND IT! MY MOTHER MUST HAVE STARTED THE FIRE WITH IT!

YEEEETCH, Gridley. We must remember that patience is a virtue.
- Pin 25 = A0, pin 27 = A1, pin 40 = A2, pin 34 = A3,
- pin 31 = A4, pin 35 = A5, pin 38 = A6, pin 36 = A7.

VAR/80 'OUTPUT' DECODING LOGIC:

Each of the eight latching VAR/80 DBO (data bus outputs) are turned 'ON' whenever the binary value of the OUT0,xxx instruction contains a '1' that = its position in the DBO row. We will use PORT zero for illustration for the rest of this section. Forinstance:

xxx = BINARY	DBO 'ON'	xxx = BINARY	DBO 'ON'
1 = 00000001	0	2 = 00000010	1
4 = 00000100	2	8 = 00001000	3
16 = 00010000	4	32 = 00100000	5
64 = 01000000	6	128 = 10000000	7

If we wanted to turn BOTH relays 'ON' that are connected to DBO terminals zero and one, we would OUT0,3 since 3 = binary 00000011. If we wanted to turn all the outputs on at once, we OUT0,255 since 255 = binary 11111111.

REMEMBER GRIDLEY:

- DBO positions 0 and 1 have SPDT (single-pole double-throw) relays capable of handling 110-120 volts ac at 3 amps with BOTH normally OPEN and normally CLOSED contacts available.

- BOTH DBO zero and one relays have individual red LEDs to indicate if they have been activated.

- DBO positions 2 through 7 have standard TTL outputs which equal a nominal 5 volts dc output when ON and a nominal zero volts dc output when OFF.

- ALL DBO outputs are automatically 'LATCHED.' That is, each output will remain the same UNTIL another OUT0,xx statement comes along that changes its status. To RESET the latches simply include the statement OUT0,0.

- The status of DBO positions 2 through 7 may be tested by connecting a wire to the top TEST terminal in the DBO terminal strip row and touching this wire to the DBO output terminal. IF the red TEST LED lights, it is ON. Otherwise it is OFF. If you wish, you may add a red LED for each DBO output from 2 to 7 just to the left of the terminal strip. Be sure to include a series 2K ohm resistor with each one to ground. The TTL output is adequate to drive it.

VAR/80 INPUT DECODING LOGIC:

In BASIC, try the following on any PORT except '255' which is latched at 127 by flip-flop Z24.

PRINT INP(xxx)....then ENTER.

ALL (except 255) PORTS will display 255 = 11111111 binary.

THEN THAT MEANS THAT 'ALL' INP's ARE FULLY ACTIVATED ? ? ? OUT0,255 ACTIVATED 'ALL' THE VAR/80 DBO'S ! ! !

Not exactly, Gridley. PORT input value decoding for the TRS-80 and VAR/80 is EXACTLY the opposite of PORT output value logic.

YOU MEAN IT IS BACKWARDS ? ? ?

Backwards is a bit too strong, Gridley. Better we should say it is 'inverted.' All we mean is that all 1's = zero and all zeros = 1. The easiest way to invert the INP(xxx) value to decimal is to simply subtract the xxx value from 255.

YOU MEAN IF X = INP(0) & X = 255 THEN 'X' REALLY = ZERO ? ? ?

Verrrry good, Gridley. You've got it tooth and nail...hammer and tong, down pat. The VAR/80 input terminal strip labeled DBI (data bus input) numbers two through seven, are normally held at a logic 1 = nominal 5 volts dc (actually TTL logic 1 is defined as any dc voltage from +2.4 to +5.0 volts and TTL logic zero as any voltage from zero to +1.7 volts dc...+1.7 to +2.4 volts dc is "NO MAN'S LAND = DON'T CARE").

DBI TERMINALS ZERO & ONE = OPTICAL-COUPLER ISOLATED INPUTS:

'Opto-couplers' are a neat invention that goes back over 100 years. They have been developed in their present form specifically to PRESERVE the very tender gates in most all TTL circuitry from abuse that results in their acting like fuses whenever a voltage greater than +5 volts and/or a current greater than a few milliamps is run through their input. They consist of nothing more than an LED, a dropping resistor to limit the LED current and most importantly a PHYSICALLY SEPARATED light sensitive transistor which discerns whether the LED is ON or OFF and outputs an appropriate TTL signal. Figure 5-5 illustrates how the 2 opto-couplers on DBI terminals 0 and 1 operate.

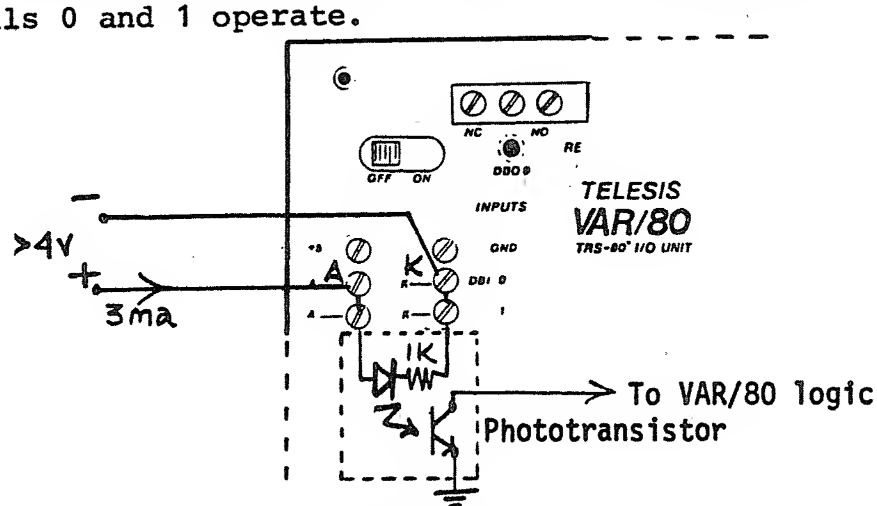


Figure 5-5

The DBI (data bus input) terminals numbers 2 to 7 are held at a nominal +5 volts dc when NO signal is input to them. They are activated by grounding them, and/or a TTL logic zero. Terminals zero and 1 with the opto-couplers, require a minimum of +4 volts dc on the 'A' anode side with the 'K' cathode side grounded, to turn-on the built-in LED. When this LED comes 'on', then the phototransistor goes low, a TTL logic zero, thus activating its DBI, either DBI 0 or 1. Opto-couplers are a neat way of protecting the TTL circuitry from being "electrocuted" by your external actuating circuit, whatever it might be. Actually, the 'A' anode side will handle up to 20 volts dc without BLOWING its LED. Plus 5 volts dc is optimum.

The VAR/80 very conveniently provides +5 volts dc at the terminal just above the two 'A' connections. As such, DBI inputs zero and 1 can be activated exactly the same as inputs 2 through 7, by grounding them. Figure 5-6 shows the hookup.

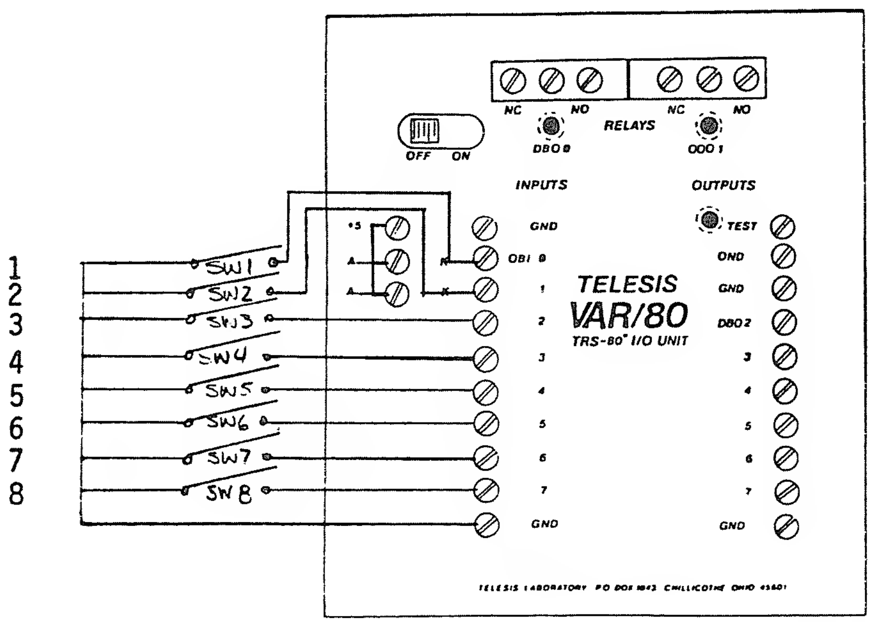


Figure 5-6

The last 3 pages of this Chapter illustrate additional ways of driving the DBI inputs, plus useful circuits for coupling the DBO outputs to: an NPN xstr driven lamp or relay, NE555 audio oscillator, CMOS, and TTL circuits; all courtesy of Telesis.

SOME FUNDAMENTAL BASIC/ASSEMBLY LANGUAGE PGMS FOR THE VAR/80:

Let's start off with a BASIC program that monitors the 8 DBI inputs for being grounded as shown in Figure 5-6's hookup.

DBI INPUT TEST PROGRAM - BASIC:

```
10 CLS:E=256
20 D=INP(0):IFD=EGOTO20ELSECLS
30 E=D:I=1:FORX=0TO7:Q=DANDI
40 PRINT"DATA BUS LINE #D";X;"= ";
50 IFQ>0PRINT"OFF"ELSEPRINT"ON"
60 I=I*2:NEXTX:A=A+1:IFA=1GOTO20
70 PRINT:PRINT"SOMEBODY RANG MY BELL";:INPUTR:A=0
80 D=INP(0):IFD<255GOTO90ELSE20
90 PRINT@832,"BELL STILL RINGING":GOTO80

RUN

DATA BUS LINE #D 0 = OFF
DATA BUS LINE #D 1 = OFF
DATA BUS LINE #D 2 = OFF
DATA BUS LINE #D 3 = OFF
DATA BUS LINE #D 4 = OFF
DATA BUS LINE #D 5 = OFF
DATA BUS LINE #D 6 = OFF
DATA BUS LINE #D 7 = OFF
BREAK IN 20
READY
>_
```

WHAT HAVE WE GOT HERE ? ? ?

Not very much. Just a little 9 liner BASIC program that tests PORT ZERO's input value and prints out on video the status of the switch closures on DBI terminal strips zero through 7. IF any one or combination of switches is closed, it will signal the fact on the video display with an 'ON' for the appropriate terminal or terminals, and also 'LATCH' the input ....plus printing out on video, SOMEBODY RANG MY BELL. REMEMBER, the data bus inputs are NOT latched, so it is convenient to have the program (when appropriate) do the latching. You may reset the latch/latches by pressing any key on the keyboard and then ENTER.

IF all the switches are clear (open) the program awaits the next switch closing. If NOT clear, the program prints out on video, BELL STILL RINGING.

Mr. V.A.R. wrote the fundamental program. 'Thank you.' We only modified it a tiny 'bit.' Here is how it works:

10: Variable E is intialized at any value greater than 255 so so that line 20's IF statement sends us to line 30 the FIRST time after RUN.

20: Variable D is assigned the value on the data bus for PORT zero and IF equal to E (meaning nothing has changed from the first pass through the program) then loops back for another look at the data bus via PORT zero. IF it has changed, then CLS and proceed down to line 30.

30: Variable E is assigned D's value for the test in line 20. Variable I is set at one since I will be AND'ed with D in the 3rd statement in this line, PLUS being multiplied by 2 in line 60. As such, each time this line's FOR statement loops through NEXT in line 60, I will = 2, 4, 8, 16, 32, 64, & 128. This allows the program to TEST the value of each bit sequentially and individually, somewhat like the assembly language BIT instruction which places the complement of the tested bit's value in the Flag register.

40: The program prints out on video 'DATA BUS LINE #' and X which = 0 through 7, and the equals sign.

50: Prints out OFF if the ANDing of D and I in line 20 were greater than zero, ELSE prints out ON.

60: Multiplies I * 2 to test the next bit on the data bus, furnishes the NEXT for line 30's FOR, increments A and IF A = 1 then goes back to line 20 or IF not = 1, then drops down to line 70.

70: This line prints out the BELL message and effectively latches the switch closure till you hit any key and then ENTER. Variable A is reset to zero before the program drops down to line 80.

80: Tests the value of the data bus via PORT zero and IF the data bus is NOT clear, <255 then jumps to line 90, and IF clear, then jumps to line to for another look.

90: IF the data bus is NOT clear, this line informs us of the fact and continues looping back to line 90 UNITL it is actually cleared.....at which time line 20 begins the entire procedure all over again.

This little 9 liner BASIC routine is not a very sophisticated program, but it does illustrate three important points:

1. Using BASIC's AND instruction to check the value of each bit in the data bus' byte from PORT zero and inform us of its value by multiplying I * 2 every time it loops.

2. Using BASIC to 'latch' the value of the data bus input IF it is NOT 255 = 111111111 binary = "nothing is happening."

3. NOT having to use SPECIFICALLY annotated PRINT@ video addresses for each DBI terminal (0 to 7) by using CLS.

There are all sorts of ways to write a BASIC program that does exactly the same thing, Gridley. REMEMBER that for INPut:

DECIMAL	BINARY	DBI NO.	DECIMAL	BINARY	DBI NO.
254	11111110	0	253	11111101	1
251	11111011	2	247	11110111	3
239	11101111	4	223	11011111	5
191	10111111	6	127	01111111	7

WELL, THAT CHART ON THE BOTTOM OF THE LAST PAGE STILL LOOKS 'BACKWARDS' TO ME, WHEN COMPARED WITH OUT PORT ZERO, VALUES! !

Very well, Gridley. Try looking at it through the LOOKING GLASS that Alice used. Maybe all the zeros will look like ones and vice versa. Does that make you more comfortable ? ?

ARE YOU SUGGESTING I'M LIKE THE MARCH HARE ? ?

No, Gridley. Certainly not, but you do resemble a certain HATTER we all fondly recall. Now, let's get down to business and write a program in assembly language that is similar to our last BASIC program. We'll write it for the VAR/80 and use PORT zero to illustrate how it works. Get out your dictionary for modern "FAGGINSYLVANIAN" and pay attention, Gridley.

INPUT PORT ZERO DEMONSTRATION PROGRAM IN ASSEMBLY LANGUAGE:

Is on the following 2 pages. Figure 5-7 is the source code and Figure 5-8 the object code for this program. The program's output to the video display is identical to the earlier BASIC program, but the BELLS and whistles have been left out for the sake of brevity. If you choose to load the program from DOS with NEWDOS+ or NEWDOS80, remember to protect memory by using: BASIC,32000 then ENTER. In disk BASIC, all you need do is type in SYSTEM and then ENTER, and /32000 then ENTER to get the program up and running. The program RUNs equally well in non-disk Level 2 BASIC and standard 2.2 or 2.3 disk BASIC, your choice. CAUTION: Some of the very early (circa 1977 & 1978) TRS-80s will NOT tolerate the VAR/80 being plugged into the screen printer port while running 4 disks, etc., since some of the data and address lines will not handle the extra loading. Simply unplug it till you have loaded the program and plug it back in again when in SYSTEM and ready to run.

Now let's take a look at the source code program. There is certainly nothing mysterious or difficult about following the program's logic or flow IF you will read the comments for each line, but for the sake of newcomers and latecomers, and you too Gridley, we'll run through it line by line and try to amplify some of the comments. A few of the Z-80 instructions, such as BIT and RRCA have NOT been used in programs in either Volumes 1 or 2 of the Disassembled Handbook For TRS-80, so the following amplified comments MAY be of assistance.

120: ORiGin is at 32000 decimal = 7D00 hex for those of you with 8 fingers on each hand. The program is easily relocated IF you change lines 140 and 150 appropriately. We will use decimal for most instructions, UNLESS hex is more convenient. SOME memory locations like 032AH to display on video the 'A' register and 0DH for a carriage return = skip-a-line-on-video are more easily memorized in hex than decimal. Nevertheless, we will TRY to use decimal MOST of the time. Remember, any value with an 'xxx' asterisk on each side is interpreted by the assembler as an ASCII value.

```
00100 ; VAR/80 INPUT DEMO - 128 BYTES - OBJECT=INP1/SOURCE=INP2
00110 ;
00120          ORG     32000            ;LET'S START THE PGM HERE
00130          CALL    01C9H            ;ROM CLS SUBROUTINE
00140 PORT0    EQU     32200            ;PORT ZERO VALUE STASH
00150 COUNT    EQU     32201            ;LINE COUNTER STASH
00160 AGAIN    IN      A,(0)            ;LD 'A' PORT ZERO VALUE
00170          LD      (PORT0),A        ;STASH IT AWAY AT 32100
00180          LD      A,48             ;48 = ASCII 'ZERO'
00190          LD      (COUNT),A        ;ZERO OUT COUNTER STASH
00200 TELL     LD      HL,SHOW          ;MESSAGE STRING MEM ADDR
00210          CALL    28A7H            ;DISPLAY STRING ON VIDEO
00220 SHOW     DEFM    'DATA BUS LINE #D '
00230          DEFB    00               ;END OF MESSAGE DELIMITER
00240          LD      A,(COUNT)        ;COUNTER STASH TO 'A'
00250          INC     A                ;ADD +1 TO 'A' REGISTER
00260          LD      (COUNT),A        ;STASH IT AWAY AT COUNT
00270          DEC     A                ;SUBTRACT -1 FROM 'A'
00280          CALL    032AH            ;DISPLAY 'A' ON VIDEO
00290          LD      HL,EQUAL         ;EQUAL MESSAGE LOCATION
00300 EQUAL    DEFM    ' = '            ;EQUAL ASCII MESSAGE
00310          DEFB    00               ;DEFM MESSAGE DELIMITER
00320          CALL    28A7H            ;DISPLAY $ ON VIDEO
00330          LD      A,(PORT0)        ;PORT0 VALUE STASH TO 'A'
00340          BIT     0,A              ;TEST BIT 0 SET Z COMPLE.
00350          JR      Z,ONAGN          ;GOTO 'ONAGN' IF ZERO
00360          LD      HL,OFFMES        ;OFF MESSAGE ADDRESS
00370 OFFMES   DEFM    'OFF'            ;'OFF' MESSAGE
00380          DEFB    00               ;MESSAGE DELIMITER
00390          CALL    28A7H            ;DISPLAY IT ON VIDEO
00400          LD      A,(PORT0)        ;MEM STASH VALUE TO 'A'
00410          RRCA                     ;ROTATE 'A' 1 BIT RIGHT
00420          LD      (PORT0),A        ;UPDATE MEM STASH
00430 FINIS    LD      A,0DH            ;0DH=GOTO NEXT VIDEO LINE
00440          CALL    032AH            ;DO IT ON VIDEO
00450          LD      A,(COUNT)        ;COUNT VALUE TO 'A' REG
00460          CP      56               ;-56 FM 'A' SET Z FLAG
00470          JP      NZ,TELL          ;GOTO TELL IF NOT ZERO
00480 LOOP     IN      A,(0)            ;PORT 0 VALUE TO 'A'
00490          LD      HL,PORT0         ;PORT0 MEM ADDRESS STASH
00500          CP      (HL)             ;SUB MEM CONTENTS FM 'A'
00510          JR      Z,LOOP           ;GOTO LOOP IF ZERO
00520          CALL    01C9H            ;ROM CLS SUBROUTINE
00530          JP      AGAIN            ;START ALL OVER 'AGAIN'
00540 ONAGN    LD      A,'O'            ;ASCII 'O' TO A
00550          CALL    032AH            ;DISPLAY 'O' ON VIDEO
00560          LD      A,'N'            ;ASCII 'N' TO A
00570          CALL    032AH            ;DISPLAY 'N' ON VIDEO
00580          LD      A,(PORT0)        ;MEM STASH TO 'A' REG
00590          RRCA                     ;ROTATE RIGHT 1 BIT
00600          LD      (PORT0),A        ;UPDATE MEM STASH
00610          JP      FINIS            ;GOTO FINIS
00620          END     32000            ;EL FIN = EL BEGUINE
```

Figure 5-7 Source Code

```
7D00            00120          ORG     32000
7D00 CDC901     00130          CALL    01C9H
7DC8            00140 PORTO    EQU     32200
7DC9            00150 COUNT    EQU     32201
7D03 DB00       00160 AGAIN    IN      A,(0)
7D05 32C87D     00170          LD      (PORTO),A
7D08 3E30       00180          LD      A,48
7D0A 32C97D     00190          LD      (COUNT),A
7D0D 21137D     00200 TELL     LD      HL,SHOW
7D10 CDA728     00210          CALL    28A7H
7D13 44         00220 SHOW     DEFM    'DATA BUS LINE #D '
7D24 00         00230          DEFB    00
7D25 3AC97D     00240          LD      A,(COUNT)
7D28 3C         00250          INC     A
7D29 32C97D     00260          LD      (COUNT),A
7D2C 3D         00270          DEC     A
7D2D CD2A03     00280          CALL    032AH
7D30 21337D     00290          LD      HL,EQUAL
7D33 20         00300 EQUAL    DEFM    ' = '
7D36 00         00310          DEFB    00
7D37 CDA728     00320          CALL    28A7H
7D3A 3AC87D     00330          LD      A,(PORTO)
7D3D CB47       00340          BIT     0,A
7D3F 282C       00350          JR      Z,ONAGN
7D41 21447D     00360          LD      HL,OFFMES
7D44 4F         00370 OFFMES   DEFM    'OFF'
7D47 00         00380          DEFB    00
7D48 CDA728     00390          CALL    28A7H
7D4B 3AC87D     00400          LD      A,(PORTO)
7D4E 0F         00410          RRCA
7D4F 32C87D     00420          LD      (PORTO),A
7D52 3E0D       00430 FINIS    LD      A,0DH
7D54 CD2A03     00440          CALL    032AH
7D57 3AC97D     00450          LD      A,(COUNT)
7D5A FE38       00460          CP      56
7D5C C20D7D     00470          JP      NZ,TELL
7D5F DB00       00480 LOOP     IN      A,(0)
7D61 21C87D     00490          LD      HL,PORTO
7D64 BE         00500          CP      (HL)
7D65 28F8       00510          JR      Z,LOOP
7D67 CDC901     00520          CALL    01C9H
7D6A C3037D     00530          JP      AGAIN
7D6D 3E4F       00540 ONAGN    LD      A,'O'
7D6F CD2A03     00550          CALL    032AH
7D72 3E4E       00560          LD      A,'N'
7D74 CD2A03     00570          CALL    032AH
7D77 3AC87D     00580          LD      A,(PORTO)
7D7A 0F         00590          RRCA
7D7B 32C87D     00600          LD      (PORTO),A
7D7E C3527D     00610          JP      FINIS
7D00            00620          END     32000
00000 TOTAL ERRORS
```

Figure 5-8 Object Code

130: Here is our old friend in ROM that very neatly CLS.

140: PORT zero is the label for the location in MEM where the program will stash away the intial value of Port zero, and its UPDATED value AFTER the RRCA instruction rotates it ONE bit to the right....more on this later.

WHY USE 'EQU 32200' INSTEAD OF 'PORTO DEFB 0.' ? ? ?

A good question, Gridley. It is strictly a matter of choice, said the Dutch milkmaid as she kissed the cow. DEFB will work just as well. We choose to use a MEM location that is EASY to remember for trouble shooting the program IF any glitches occur. It is a lot easier FOR US to switch back to BASIC via the JP 0072H instruction and PEEK(32200) than to load T-BUG or Z-BUG or whatever 'trouble shooting' program you fancy. Use whatever is EASIEST and most convenient for YOU. There is no RIGHT or WRONG approach as long as it WORKS for you.

150: This MEM location does TWO jobs for us. It keeps track of the number of times (eight) we have run through the PORTO byte to decode EACH bit, plus by intializing it at ASCII zero, very conveniently allows the program to print out the DBI bit number, 0 through 7, WITHOUT having to ADD 48.

160: IN A,(0) = INP(0) in BASIC with PORT zero's value to 'A.'

170: The program stores the INITIAL value of IN A,(0) @ 32200.

180: As mentioned above, we'll initialize our bit counter at ASCII 48 = zero.

190: And stash it away in MEM location 32201.

200: HL register is loaded with the MEM address of our message in line 220.

210: CALL 28A7 hex is another old friend in ROM that displays the message in RAM beginning at HL and terminated with a zero, IF MEM location 409CH contains a 'zero.' Fortunately, 409CH is intialized at 'zero' (+1 = LPRINT output and -1 = cassette output for 28A7H), so we do not need to bother about it.

220: This is the message CALL 28A7H will output to video. The apostrophes at each end tell the assembler to store it in ASCII format.

230: The DEFB zero tells our ROM's CALL 28A7H display a string subroutine, that THIS IS THE END of the message.

240: Register 'A' loaded with the value from 32201 in MEM which is our ASCII bit zero to seven MEM stash location.

250: Increment 'A' register by +1.

260: Put the updated value in 32201 MEM location.

270: Decrement 'A' by subtracting -1. WHY, Gridley ? ? ?

IT MAKES NO SENSE AT ALL TO ME. FIRST YOU INCREMENT IT AND THEN YOU DECREMENT IT. WHY BOTHER ? ? ?

Just being careful, Gridley. Caution is sometimes the better part of valor. What IF we CALLed 032AH, as we will in the next line, line 250, and this CALL modified 'A' during its execution and then we incremented 'A' and stored it in 32201? What would happen?

G-I-G-O. GARBAGE-IN-GARBAGE-OUT. OK, I UNDERSTAND.

280: Another old friend from ROM. Display the contents of 'A'.

290-320: Display ' = ' ASCII string on video.

330: Load 'A' register with the value in 32200 MEM that INITIALLY came from line 160's IN A,0.

340: Test BIT zero of the 'A' register and store its complement in the Z flag position of the 'F' register. Since the Z flag uses a 1 = ZERO and a 0 = NOT ZERO for any arithmetic or Boolean operation, then IF the bit tested was a zero, the Z flag will contain a 1. And;

350: JR,Z ONAGN will send the program off to line 540 to output an 'ON' IF the bit tested was a zero. IF totally confused, goto the bottom of page 5-21 and refresh your memory regarding the value of EACH bit when each separate DBI number is 'ON.'

360-390: IF the tested bit = 1 = 'OFF' then these 4 lines output the OFF to the video display.

400: Load 'A' register with the value stashed in 32200.

410: RRCA along with line 340's BIT opcodes are the real 'work horses' of this mini-program. RRCA's instruction = 'ROTATE ACCUMULATOR RIGHT CIRCULAR' for ONE bit, every time it is used. In the BASIC program a few pages back, we used AND 'I' to test each BIT by multiplying variable 'I' times 2 each time around. This time we will have BIT stand still and always test bit zero, and have RRCA move the byte ONE postion to the right to test each of the 8 bits. There are many ways to 'skin these cats' (sorry about that Harlequin and R/C), and this is only ONE of them. Line 420 stashes it away in MEM.
51
430-440: Label FINIS = the end and inserts a carriage return (0DH) so our DBI's are NOT all strung together on video.

450-470: Test MEM location 32201 to see IF all 8 lines and their respective bit status have been printed out and IF NOT then goes back to TELL till the job is done. When all 8 lines are printed out, it falls through to line 480. Gridley, can you suggest TWO other ways we could have stored the counter?

SURE I CAN. YOUR JUST SAID A FEW MOMENTS AGO THAT DEFB ZERO WOULD WORK JUST AS WELL. - - - - I HAVE FORGOTTEN THE 2ND WAY.

Well Gridley, you got half of it. Verrry good. Do you recall from Volume 2 that ROM never uses the IY register?

S-I-L-E-N-C-E......................THAT WAS A LONG TIME AGO!

Since ROM never uses the IY register, could we have just as well have used it? After all, we can INC and DEC IY.

THEN 'WHY' DIDN'T YOU USE IT ? ? ?

FIRST REASON: again, it is convenient to know exactly where a variable is stored for trouble shooting without having to look at the object code listing.
SECOND REASON: without a bit of manipulation that requires some MEM too, you cannot CP (compare) the IX or IY registers. This fortunately can be done with the Z8000 (son of the Z-80) whether it is an 8, 16, or 32 bit long byte. Let's stop this this not-so-fascinating discussion and finish the Chapter.

480-530: Once the program has printed out on video the status of all 8 bits representing the ON or OFF status of each of PORT zero's DBI positions on the VAR/80, these lines compare, that is subtract the old value of the data bus from the new value in a LOOP until that value changes. When it does change value, then line 520 CLS and it goes to AGAIN to start all over again. If you wish to do so, by all means add a 'latch' here, such as 'press' ENTER or whatever suits your fancy. Also, this is a good spot to insert a 'BELL' announcement whenever PORT zero's INput status changes.

EXPANDING THE VAR/80'S IN AND OUT DECODING CAPABILITIES:

May be done in exactly the same way outlined earlier in this Chapter for PORT 255, only it is a great deal easier. If you wish to add a single 4 line to 16 line demultiplexer such as the 74LS154 on DBO OUTPUT positions 4 through 7, it should be an easy job. This will give you 20 independently controllable outputs which should satisfy even the most demanding requirements. If not, use two 74LS154s = 32 controllable outputs.

There are all sorts of ways to expand your PORT zero inputs on the VAR/80. One of the easiest is to use two 74148 PRIORITY ENCODERS. These inexpensive TTL chips (about $1.29 each from Hobbyworld) allow you to encode 8 data lines to 3-line binary and may be connected directly to the VAR/80 DBI input terminals with the first one's output to terminals 2, 3, and 4 and the second one's output to terminals 5, 6, and 7. This will give you a total of 18 independent DBI input sources.

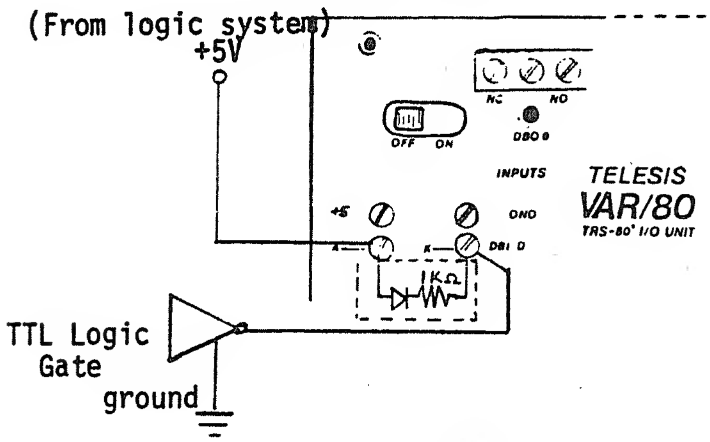
YOU SURE BEAT DIGITAL IN & OUT TO DEATH! HOW ABOUT ANALOG ? ?

Thank you, Gridley. That's what the next Chapter is all about.

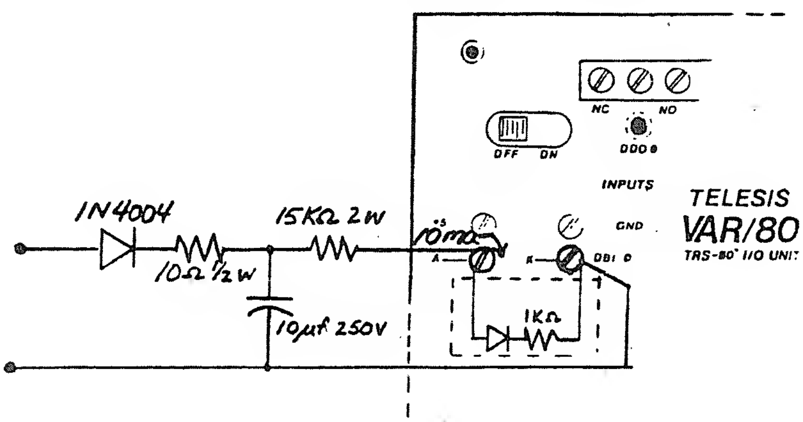
CHAPTER 5 APPENDIX:

SAMPLE DRIVE CIRCUITRY

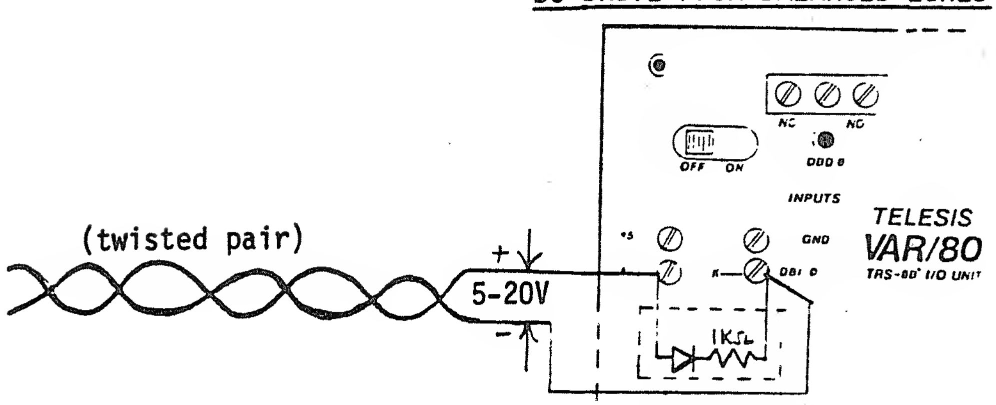
5V LOGIC DRIVE (LED ON when TTL gate is low)



115VAC LINE DRIVE

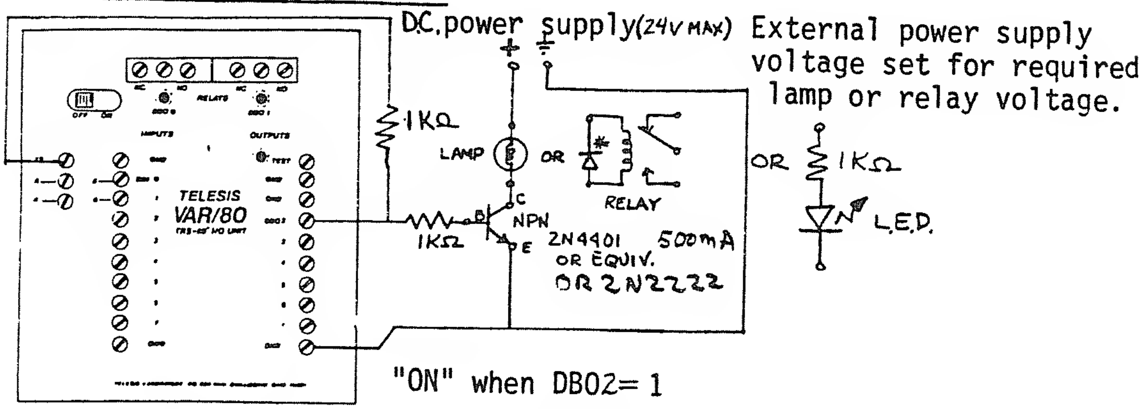


DC DRIVE FROM BALANCED LINES



CHAPTER 5 APPENDIX:

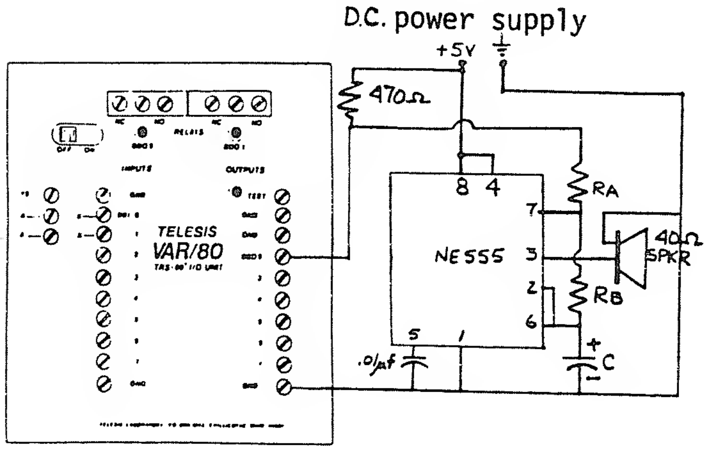
LAMP/RELAY/LED DRIVER



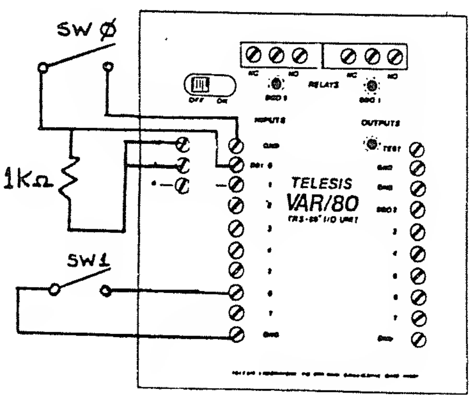
"ON" when DB02= 1

* Protective diode 1N4004 or equivalent

AUDIO ALARM



SWITCH INPUT
(opto-coupler & TTL)



$$\text{Alarm Frequency} = \frac{1.44}{(R_A + 2R_B)\,C} \quad \text{(approx)}$$

SW Ø is DBI Ø
SW 1 is DBI 6

Example:

If: $R_A = 10 \text{ Kohms} = 10^4$

$R_B = 10 \text{ Kohms} = 10^4$

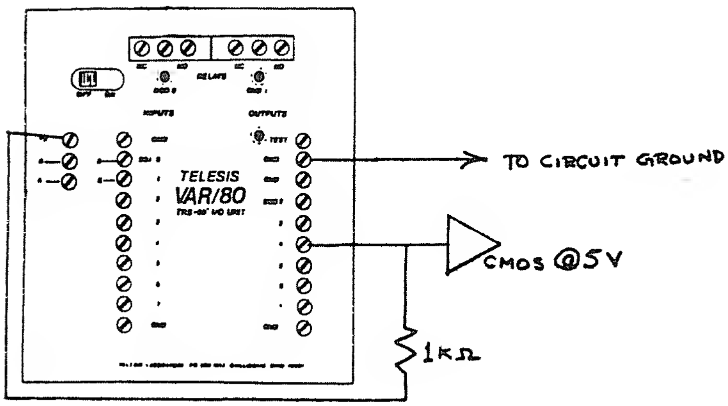
$C = .1 \times 10^{-6}$ farad (or .1 microfarad)

Then: $$\text{Freq} = \frac{1.44}{(3 \times 10^4)(.1 \times 10^{-6})}$$

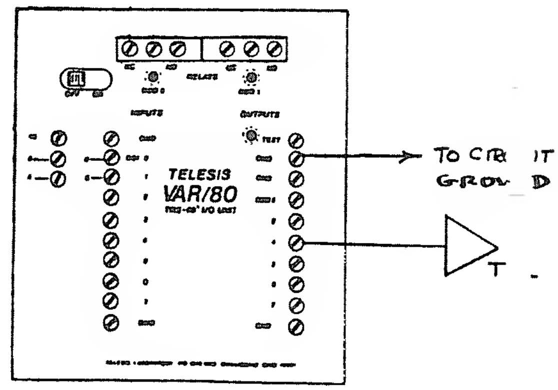
$$= \frac{144}{.3} = 480 \text{ Hz} \quad \text{(approx)}$$

CHAPTER 5 APPENDIX:

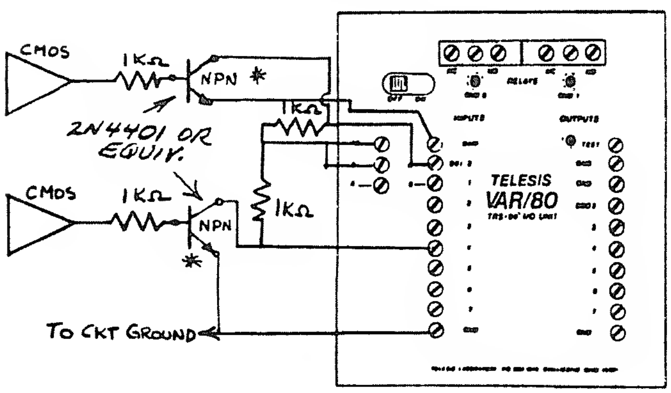
VAR/80 TO CMOS LOGIC



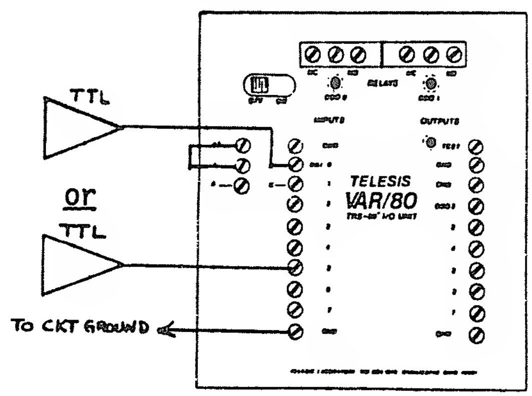
VAR/80 TO TTL LOGIC



CMOS LOGIC TO VAR/80



TTL LOGIC TO VAR/80



*Buffer transistors invert CMOS data

- CHAPTER 6 -

AN INTRODUCTION TO ANALOG TO DIGITAL CONVERTERS

INTRODUCTION:

It was only a few years ago that 'digit' meant ONLY having to do with the fingers or toes, OR with the numbers 1 through 9. 'Digital' the noun, was defined by most dictionaries as 'key of a piano, organ, accordion, etc., played with the fingers.

It was only a few years ago that the electrical/electronic engineering student was taught subjects that ONLY dealt with ANALOG voltages and currents. The ONLY digital device in those days was a 'switch' which one turned on or off with the fingers; be it a light or power switch, or Morse code key.

My goodness, how technology has changed in the last 10 years or so. Your TRS-80's keyboard unit has only TWO significant analog devices in it. Both of them are voltage regulators. The hundreds, actually thousands of digital gates in the keyboard unit, including the Z-80 microprocessor, are ALL digital devices of one sort or another.

The switch from analog to digital devices is not quite as all encompassing as the last paragraph implies. One does not take a 'Rip Van Winkle' decade long sleep from 1970 to 1980 and upon awakening find EVERYTHING changed from analog to digital.

The analog to digital change is much like our ongoing change to the metric system.....it will happen in due course, but no one is rushing it any more than necessary. In the meantime, we of necessity might as well learn how to cope with that largely STILL analog world out there beyond the edge of our TRS-80's desk full of digital goodies.

How do we interface our TRS-80 to a simple analog thermocouple to measure temperature? Or a strain gauge to tell us when the sky is falling in? Or a pressure gauge with analog output that may tell us when our Stanley Steamer's boiler is about to explode? Or an analog output fuel gauge that is warning us that our aircraft is about to run its number one tank dry?

Admittedly, this is theater of the absurd to put our point across. Hopefully, you will NOT put your TRS-80 in charge of any LIFE or DEATH situations. The fact nevertheless remains that a considerable part of the outside real world is by nature ANALOG and will remain so for a long time to come.

That is what this Chapter is all about.

We will cover the ways and means of INPUTTING analog signals to our TRS-80, both theory and practice, and save outputting analog signals from our TRS-80 for another chapter or volume.

A SLIGHTLY DIFFERENT APPROACH TO THE SUBJECT:

We are going to presume that Gridley has NEVER even seen an analog to digital converter, much less having previously studied the subject.

YOU ARE RIGHT, I'VE NEVER SEEN ONE BEFORE ! ! !

Verrry good, Gridley. We are going to use the empirical approach first; i.e., one based on experiment and observation. In doing so we will build and test a HOMEBREW A/D converter that may be constructed for a few dollars parts cost or virtually nothing IF you have a modestly stocked electronics junk box.

When we finish this lab project we will take a brief look at how a number of the other popular A/D converters work. In the last section of this Chapter we will go back into the lab and both analyze and operate the Alpha Product's 'Analog 80' 8 channel A/D acquisition module that is factory built and ready to plug into the TRS-80.

TYPES OF ANALOG TO DIGITAL CONVERTERS:

There are probably as many different types and varieties of A/D converters as there are (and have been) A/D designers. The most popular types include (not in order of popularity):

1. Single slope - indirect integrating type - our experiment

2. Dual slope - indirect integrating type - most panel meters

3. Triple slope - indirect integrating - slow, but accurate

4. Charge balancing - voltage to frequency converter

5. Servo/counter feedback type - counter drives D/A till equal

6. Tracking type - up/down counter driving D/A converter

7. Successive approximation - feedback type - very popular

8. Flash (parallel) type - fastest A/D converter - to 100 MHz

HISTORICAL PERSPECTIVE REGARDING CONVERTER TECHNOLOGY:

1955: Epsco's DATRAC A/D converter; tube type; $8000. 150 lbs
11 bits at 40 kHz conversion rate.

1958: First transistorized A/D converters; $5000. 10 - 15 lbs
12 bits at 40 microsecond conversion rates.

1966: Epsco's DATRAC 3 A/D converter; $1200. About 1 lb.
12 bits at 24 microsecond conversion rates.

1968: Redcor's encapsulated (the first) discrete A/D conv. 12 bits at 50 microsecond conversion rates @ $600.

1971: Analog Devices' monolithic building blocks for A/D conv 12 bits at 4 microsecond conversion rates @ $600.

1975: Datel Systems' hybrid ADC-12B A/D converter with 12 bits at 4 microsecond conversion less than $100.

1980: National Semiconductor's ADC0808/16 A/D conversion SYSTEMS on single chip including 8/16 channel multiplexeer/decoder, successive approximation conversion, and tri-state output latch for approximately $29. each. NOTE: this chip handles 16 SEPARATE analog inputs, so cost per 8 bit conversion is less than $2. per channel.

BUILDING A HOMEBREW A/D CONVERTER FOR THE TRS-80:

Is not at all difficult IF we use the ultra-simple single slope-indirect integrating type listed as number 1 on the previous page. First, let's review some physical facts of life about charging and discharging a capacitor through a resistor.

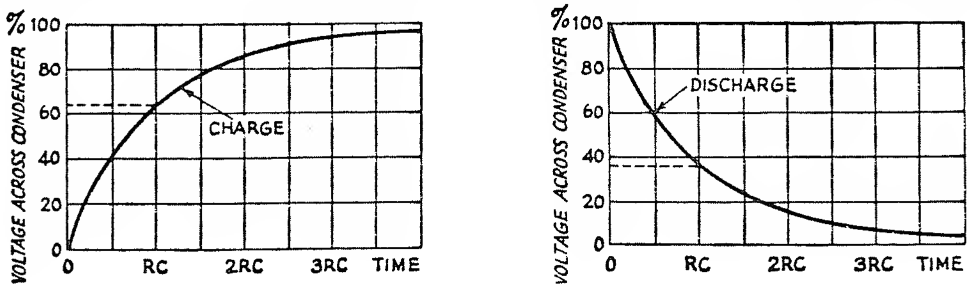


Figure 6-1 illustrates the voltage as a percentage of the source that appears across a capacitor with respect to TIME, while it is being CHARGED through a resistor R. RC is the Time constant in seconds which = R in ohms times C in farads. Time is stated in RC units. To simplify the values, we may use R = resistance in millions of ohms, and C = capacitance in microfarads. Forinstance; a 20,000 ohm resistor in series with a 50 microfarad capacitor would give us a time constant of .02 times 50 = 1 second. IF we had a voltage source of +100 volts dc, a series resistor of 20,000 ohms, and a capacitor of 50 microfarads, then Figure 6-1 shows that in 1 second the capacitor would have charged to 63 volts dc; in 2RC = 2 seconds, about 86 volts dc; and in 3RC = 3 seconds, about 95 volts dc. The relationship between the source voltage E and the capacitor voltage e may be expressed by:

$$e = E\left(1 - \ln^{-T/RC}\right)$$

Where ln is the base of the Naperian/natural logarithm raised to the -T/RC power. Fortunately, our TRS-80 uses natural logs.

Figure 6-2 illustrates the voltage that would appear across our capacitor C when it is DISCHARGED through our resistor R to ground, with respect to RC = time. Turn 6-2 upside down and it would EXACTLY fit 6-1. BOTH curves are of course, perfect Naperian logarithm/exponential curves depicting the charge of our capacitor THROUGH resistor R or discharge of our capacitor THROUGH resistor R with respect to time.

SO WHAT ELSE IS NEW ? ? ?

Well Gridley, Naperian/natural logarithms are not EXACTLY new, since John Napier in Scotland (1550-1617), invented them over 350 years ago. He not only invented natural logarithms and published the FIRST log table, Mirifici Logarithmorum Canonis, but this Highland mathematical genius also invented "Napier's Bones" (sort of an early slide rule), and MOST IMPORTANTLY introduced the DECIMAL POINT in writing numbers as we know them today. Maybe the adjective "NEW" is only a relative term when compared to the 5th century B.C. Greeks, GRIDLEY?

"S-I-L-E-N-C-E"

Very well, let's get on with building a 1980 analog to digital converter using mathematics invented in the early 1600's. Something 'old' and something 'new.' How does that grab you, Gridley.?

"KIND-A-PINK OR KIND-A-BLUE.....GUESS I'M BLUE."

You cannot win them all, Grid. Best we continue.

THE DIRECT INTEGRATING ROUTE TO CALCULATE AN ANALOG VOLTAGE:

IF we used our TRS-80 to allow an unknown voltage a specific amount of time to charge the capacitor in the Figure 6-1 curve, could we write a mini-program that would calculate the intitial voltage with a minimum of external hardware? Sure we could, BUT (a big but), it would require a VERY ACCURATE means of determining the amount TIME we allowed the unknown voltage source to charge the capacitor. For high voltages, most relays and electromechanical devices due NOT offer sufficient repeatability to both CLOSE and OPEN...2 steps to make the time measurement. So, what do you suggest, Gridley?

LET'S DO IT IN REVERSE...CHARGE IT UP...AND TIME DISCHARGE! !

Sonofagun Gridley, you are indeed an inspired inventor at times. You just 'invented' the SINGLE-SLOPE-INTEGRATING-INDIRECT analog to to digital conventer. That's EXACTLY what we will do. Using your INDIRECT approach Gridley, it makes no nevermind HOW LONG we charge up the capacitor, just so that we allow it to fully reach the level of the voltage source. The IMPORTANT part of this integrating-indirect method is to measure the TIME of discharge to zero or any predetermined level. With this method, only ONE relay movement is required

which cuts in HALF any relay movement non-repeatability error and most importantly allows us to use either a zero-crossing or pre-set TTL chip to tell our program WHEN the desired level of capacitor discharge has been reached. Figure 6-3 is an expanded illustration of Figure 6-2 with the abscissa changed from RC's to seconds since the value of our R * C = 1, and the vertical scale changed to volts dc since in this illustration we will presume a source voltage of 100. Let's ADD an upper horizontal scale that assumes we have a counter (our TRS-80) that beginning at TIME zero counts 80 counts per second till the voltage reaches +5 volts dc.

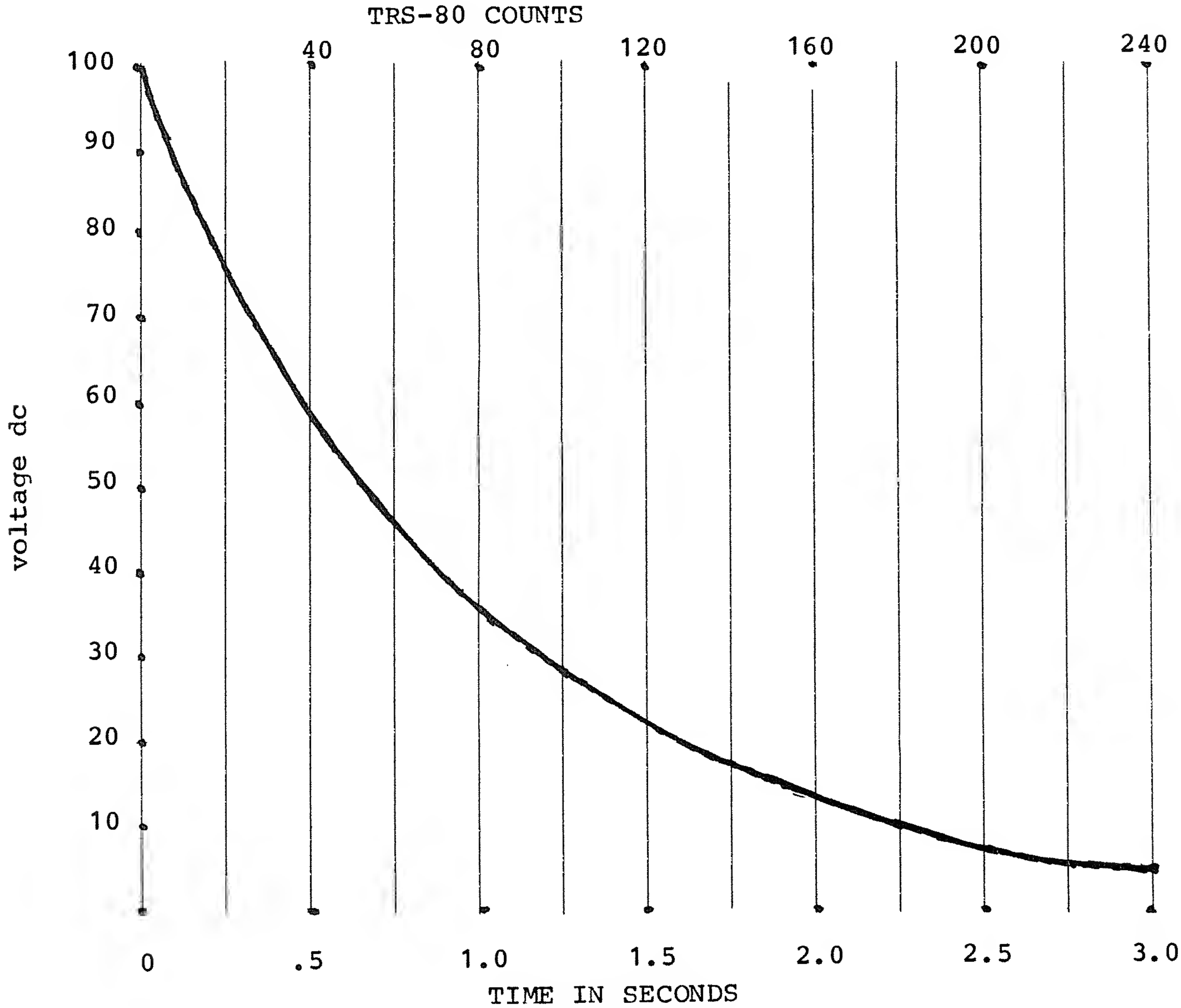


Figure 6-3

Figure 6-3 shows that IF we had +100 volts dc input across capacitor C (50 microfarads), and resistor R (20,000 ohms) was connected to ground, that it would take EXACTLY 3 seconds for the voltage to drop to a value of +5 volts dc. Solving the equation on the bottom of page 6-3 for E, assuming e = 5 volts and our TRS-80 program made 80 counts per second, THEN:

E (unknown source voltage) = 5 * (EXP(T/80))

All we have to do to turn our TRS-80 into an analog to digital converter is add a few parts and write a mini-BASIC (it could just as well be in assembly language) program to do the timing and solve the equation on the bottom of page 6-5.

IT SURE SOUNDS SIMPLE, BUT HOW DO WE DO IT ? ? ?

A good question, Gridley. We are going to follow Dave Lien's advice to KISS (keep-it-simple-Simon) and use a little poetic license with the design of our adapter circuitry to make it VERY EASY to construct. The circuit trades off a bit of accuracy for the sake of simplicity, BUT for those who wish either 1 percent or 5 percent of reading accuracy, we will include an optional error correcting IF-THEN subroutine that will compensate for offset, gain, and linearity errors that our somewhat 'less than' PERFECT A/D converter may entertain due to its somewhat 'less than' IDEAL transfer functions AND non-lab standard components; i.e., we will use junk box 20 percent or worse tolerance components. No Gridley, we will NOT require laser trimming of the resistors to 1/100th percent tolerance since you, and possibly a few readers, may not have a fully automated laser readily available.

INTERFACING THE A/D CONVERTER TO THE OUTSIDE REAL WORLD:

There are three fundamental ways of interfacing our TRS-80 to the outside real world for both OUT port and INP port functions.

1. You may use the cassette/CHR$(23) Port 255 WITHOUT modifying the keyboard in ANY way. You may use the cassette motor control relay K1 to control an external 6 volt dc buffer relay as described in Chapter 5 with the OUT255,4 instruction to turn it 'on' and the OUT255,0 instruction to turn it 'off. For inputting port 255 via INP(255) and using the cassette input line we may key 'on' (=255) or key 'off' (=127) by turning on or off a one transistor audio oscillator (code practice oscillator - Radio Shack #20-1155, #20-005, or a homebrew LM555 oscillator) with 1 volt peak to peak output in the range of 500-2000 cycles) to tell our conversion program when to 'start' and when to 'stop' counting.

2. You may install the latches and/or demultiplexers covered in Chapter 5 that use port 255 for both input and output.

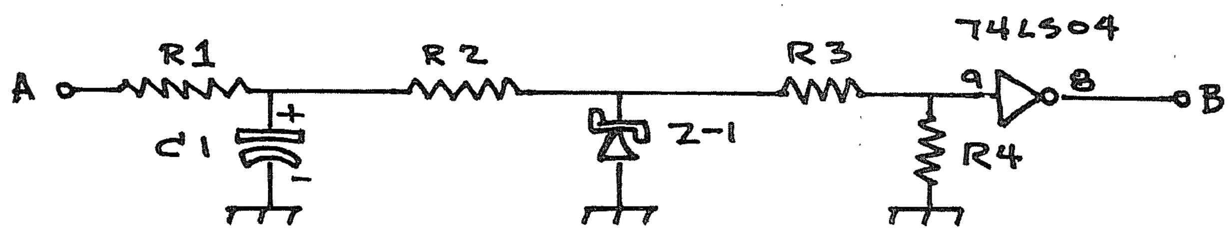
3. You may use a Telesis 'VAR/80' (as thoroughly covered in Chapter 5) or Alpha Products 'Interfacer 2' (works virtually identical to the VAR/80) to interface to port zero for both output and input instructions. This is the version we will use for the rest of this section.

You hard-core hardware buffs should have little difficultly using either method #1 or #2 above without further explanation and for the newcomers to dripping hot-solder on their trouser legs, method #3 should be the easiest one to implement.

You will recall that the VAR/80 (and the Interfacer 2) have relays with single pole, double throw, contacts rated at 3 amps @ 120 volts ac, on output positions 0 and 1. We will use the relay at position zero and turn it 'on' with the OUT0,1 instruction (remember that it latches 'on') and turn it 'off' with the OUT0,0 instruction. Have you got that, Gridley?

SURE I DO...I CAN REMEMBER 1 CHAPTER BACK & SOMETIMES MORE ! !

Verrry good, Gridley. Let's take a look at Figure 6-4's NOT too complex circuit that is the heart of our homebrew single slope, integrating-indirect A/D to converter.



R1 : 1500 ohms 1 watt
R2 : 20 K ohms 1 watt
R3 : 1500 ohms 1/2 watt
R4 : 2200 ohms 1/2 watt
14 pin DIP socket

C1 : 50 ufd @ 450 WVDC
Z1 : 4.6 VDC Zener diode
74LS04 hex buffer-inverter
(don't forget +5 VDC to pin 14 & ground pin 7.)

Figure 6-4

Most all the parts are available from Radio Shack, EXCEPT the 50 ufd at 450 working volts dc electrolytic capacitor. It may be ordered by mail from: Burstein-Applebee, 3199 Mercier St., Kansas City, Missouri 64111, stock # 15A5599-9 @ $4.06 plus postage, IF you do not have one in your junk box.

It is easily constructed in a few minutes on a piece of perf-board, Radio Shack #276-1582. Use a small #58 or #60 drill bit to drill the holes for the 14 pin 74LS04 DIP socket.

Point 'A' is connected to the VAR/80's DBO zero center relay contact 'C'. This relay's NO (normally open) contact is then connnected to your dc voltage source (any voltage from 10 to 120 volts dc). You may 'steal' the +5 VDC for the 74LS04 pin 14 from the VAR/80's front panel +5 terminal. Point 'B' is connected to the VAR/80's DBI terminal #7. Make sure the common ground connections on the perfboard and your UNKNOWN voltage source ALL are connected to one of the VAR/80 front panel GND terminals. Now, let's see how what we have built works.

R1 is an input 'surge' limiting resistor to protect the contacts on the VAR/80's DBO zero relay. C1 and R2 are our indirect-integrating RC components (.02 * 50 ufd = 1). The 4.6 VDC Zener diode Z-1 does 2 important jobs:
1. Acts as an effective short to ground till voltage = +4.6

2. Serves to protect the input gate, pin 9, on the 74LS04 buffer-inverter chip from ANY over-voltage problem; i.e., IF the voltage were to go over approximately +5 VDC, this low power Schottky gate would act like a fuse and BLOW, which could spoil your afternoon.....though with their prices so VERY low nowadays, plus the fact that you have 5 spare gates to utilize, it would only be a minor annoyance.

Resistors R3 and R4 act as a voltage divider so that (hopefully) when C1's voltage has discharged to 4.6 to 5.0 volts through R2 and Z-1, the 74LS04 buffer-inverter's OUTPUT will change from a low to high. Remember, an inverter's output is the OPPOSITE from its input. As long as bit 7 on the VAR/80 DBI terminal strip is 'low' that INP(0) = 127. When bit 7 goes 'high' then INP(0) = 255.

All our BASIC (or assembly language) program need do to have the TRS-80 calculate and print out on video the unknown analog input voltage somewhere between +10 and +120 volts dc is:

1. Charge up capacitor C1 for a second or two via an OUT0,1 statement which closes relay DBO zero.

2. Open relay DBO zero and starting counting while almost instantaneously testing INP(0). IF port zero is = 127, then keep counting. IF port zero NOT = 127, then stop counting.

3. Calculate the unknown voltage with the equation below where T = the number of counts our BASIC IF-THEN statement made.

E (unknown voltage) = 5 * (EXP(T/80))

HERE IS THE BASIC PROGRAM TO 'DO' IT:

```
10 T=0:DEFINTA-Z:OUT0,1
20 FORX=1TO600:NEXT:OUT0,0
30 IFINP(0)=127THENT=T+1:GOTO30
40 E=5*(EXP(T/80))
50 PRINT"THE D.C. VOLTAGE WAS +";E
60 INPUT"DO YOU WISH ANOTHER READING";Y:GOTO10
```

PROGRAM SUMMARY:

LINE 10: intialize T at zero; DEFINT to speed up; turn 'on' relay DBO zero to charge up C1 to unknown voltage.

LINE 20: delay about 1 1/2 seconds to fully charge C1 and then turn 'off' relay DBO zero.

LINE 30: if input port zero = 127 then increment T and go back for another look. Keep looping till input port zero NOT = 127, then drop down to line 40.

LINE 40: calculate the value of the unknown voltage 'E' using T divided by 80 which = seconds it took the capacitor C1 to discharge to approximately + 5 VDC.

LINE 50: print out the 'good news' (hopefully), on the video display.

LINE 60: press ENTER if you wish to try your luck again.

THAT WAS EVEN EASIER THAN YOU SAID IT WAS ! ! !

Thank you, Gridley. There is certainly NOTHING mysterious, difficult, or complicated about this A/D conversion system, BUT unfortunately we DO HAVE A PROBLEM; i.e., its accuracy leaves a bit to be desired. You cannot make a silk purse from a sow's ear, the old philosopher said.....or can we? Figure 6-5 is a graph of actual voltage input versus video printout over the range of +10 to +100 volts dc. The solid line is what a PERFECT A/D converter would output, and the dotted line is actually what the system output WITH OUR COMPONENTS.

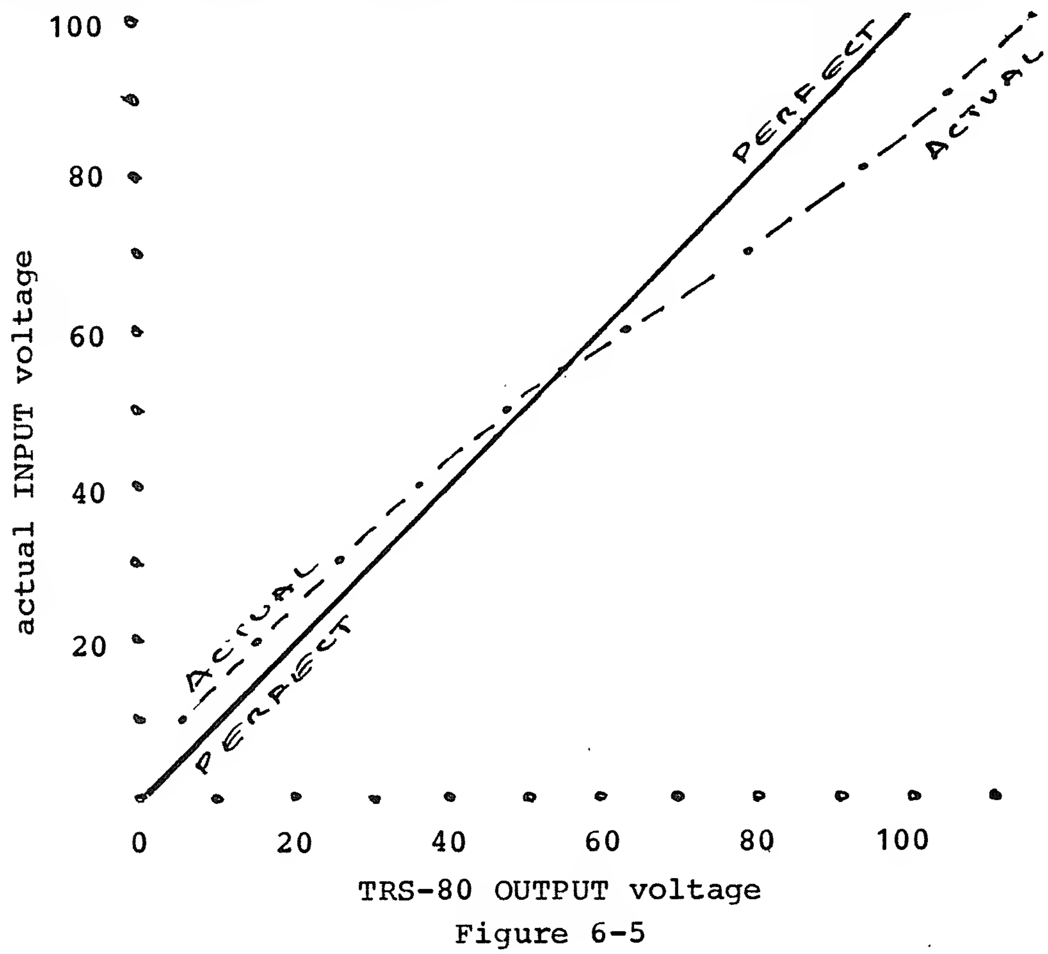


Figure 6-5

You may have been pleasantly surprised to find that your 'kluuge' of 20 percent tolerance parts was MUCH MORE ACCURATE than ours. If so, you were VERY LUCKY. It is possible for a few of the tolerance errors (some high and some low) to cancel EACH OTHER OUT. Conversely, they could be additive and give you a truly horrendous reading. A bit later, we will modify our program with old reliable IF-THEN to give us a PERFECT readout. First, let's review some of the systemic errors common to A/D converters & see IF they match-up with Figure 6-5.

Real life A/D converters do not have perfect transfer functions, especially when using electrolytic capacitors (that may be 25 years old) and 20 percent tolerance parts. We did indeed 'CHEAT' a bit with our converter by trimming, that is adjusting the values of R3 and R4, so that at the mid-range input voltage of +55 volts dc we had a perfect readout. Three errors are present in all A/D converters, though not always noticeable or objectionable in the better quality units. They do change with component ageing and temperature to varying degrees.

1. Non-linearity error which is the maximum deviation of the actual transfer function from Figure 6-5's PERFECT straight line at ANY point on the line. Our system has beaucoup of this type of error at most any point above and below +55 VDC.

2. Offset error which is the analog error caused by the transfer function NOT passing through zero. It USUALLY is relatively constant throughout an A/D converters range and may be positive or negative. Sometimes, it is easily corrected by the addition or subtraction of the offset constant.

3. Scale factor error which is the difference in the slope between actual A/D output and the PERFECT line shown in Figure 6-5. Here we have both positive and negative scale factor errors depending on which side of +55 VDC we make the measurement.

METHODS TO CORRECT A/D CONVERTER ERRORS:

Are limited only by your imagination. The easiest way is to buy or build a better grade converter. Another approach is to derive an equation that reflects the departure of YOUR converter's output from the PERFECT slope and apply the correction in your input processing (the program). Yet another approach which is by FAR the simplest, is to add a number of IF-THEN statements to correct for your converter's errors and output the correct value. This is easiest to accomplish when you are interested in a relatively narrow band of voltages, forinstance +45 to +65 volts, but may be used throughout the full range depending on your patience for writing IF-THEN statements.

One of our applications where we used this particular Mickey Mouse A/D converter was an interrupt called subroutine that monitored the thermostatically controlled temperature of a remote Gunnplexer microwave TV relay link housing. Whenever the outside air temperature fell below 20 degrees F. it was necessary to switch in an additional heating/power transistor and monitor the housing temperature a few minutes to assure that it remained stable. It just so happened that a twisted pair telephone line brought the housing's inside temperature down the mountain with +55 VDC = 120 degrees which was normal. The particular temperature sensing/thermistor system utilized had linear voltage/temperature output over the range of 110 to

130 degrees F.; i.e., 1 volt/degree. As such, 110 degrees = + 45 VDC, 111 degrees = 46 VDC, etc., on up to 130 degrees = +65 VDC. Modifying the mini-program on page 6-8 to correct for A/D conversion errors AND to print out temperature on the video display, we used the following program to monitor temperature for a few cold winter days until the modified temperature control system's reliability was established. It was crude, BUT most importantly, it worked.

```
10 CLS:T=0:DEFINTA-Z:OUT0,1
20 FORX=1TO600:NEXT:OUT0,0
30 IFINP(0)=127THENT=T+1:GOTO30
40 E=5*(EXP(T/80))
50 IFE<30THENE=E+4:GOTO170
60 IFE<38THENE=E+3:GOTO170
70 IFE<46THENE=E+2:GOTO170
80 IFE<52THENE=E+1:GOTO170
90 IFE<56THENGOTO:170
100 IFE<58THENE=E-1:GOTO170
110 IFE<60THENE=E-2:GOTO170
120 IFE<62THENE=E-3:GOTO170
130 IFE<64THENE=E-4:GOTO170
140 IFE<66THENE=E-5:GOTO170
150 IFE<68THENE=E-6:GOTO170
160 IFE<70THENE=E-7:GOTO170
170 PRINT"GUNNPLEXER HOUSING TEMP = ";E+65;"F."
180 FORX=1TO1200:NEXT:GOTO10
```

There certainly is NOTHING sophisticated about this BASIC conversion program. If you wish to read out corrected voltage instead of temperature, change line 170 to read:

```
170 PRINT"THE D.C. VOLTAGE WAS + ";E;"VOLTS"
```

Line 180's FOR-NEXT loop pauses about 3 seconds before taking another reading. Change it as desired for your application. Should you wish to expand the corrected reading range, just add additional IF-THEN statements to cover the range desired. One note of caution IF you use any extensive EXTERNAL wiring: use good earth grounds and fuse ANY inputs to to your VAR/80 or TRS-80 with 100 milliamp fuses, preferably of the automotive fast-blow (NOT slow-blow) variety as lightning strikes DO NOT aid and abet low power Schottky TTL gates in any way whatsoever. Even on Bell Tel leased phone lines, we have measured voltage spikes in the 1000 volt region. A word to the wise is sufficient and can save considerable grief.

If you MUST interface to the outside real world, we mean the outdoor world where lightning can strike and power lines can fall on phone lines, YOU can probably make a homebrew voltage surge arrestor as good as any you can buy in a store. Using a piece of scrap plexiglass, drill and mount two #10 brass bolts about one inch apart. With extra nuts, tie on two pieces of #14 copper wire with ends filed flat. Space flat ends about 2/1000ths of an inch apart. Ground one end and other to incoming phone line. Should 'arc' at about 140 VDC.

- A BRIEF LOOK AT A NUMBER OF OTHER TYPES OF A/D CONVERTERS -

DUAL SLOPE - INDIRECT INTEGRATING TYPE:

Is quite similar to the single slope we are already familiar with, BUT offers two significant improvements:

1. Conversion accuracy is INDEPENDENT of the stability of the integrating capacitor as long as it remains constant during the conversion period.

2. Conversion accuracy is INDEPENDENT of the stability of the clock counter as long as it remains constant during the conversion period.

It utilizes a built-in negative voltage reference which along with integrator linearity are the major factors that determine its accuracy.

Conversion is initiated when the INPUT voltage is switched to the integrator. Simultaneously, the INTERNAL counter starts counting clock cycles and continues counting up to overflow. The system then switches the integrator to the negative reference voltage which is then sort-of de-integrated by counting clock cycles until a comparator recognizes the zero-crossing and turns the counter 'off.' The counter output is then internally converted to a digital word that is proportional to the ratio of the 'dual slope' counts, T1 and T2.

Viirtually all panel mounted digital voltmeters and low cost digital multimeters use the dual slope system. Possibly 85 percent of all A/D converters now extant use this conversion system which offers the advantages of relatively low cost with remarkably good accuracy.

Its only significant disadvantage is that it is relatively SLOW in that it can only complete a FEW conversions per second. This apparent disadvantage can be a decided PLUS whenever measurements must be made in a noisy environment as the LONGER integating time period totally ignores the noise voltage spikes as long as they are equal to or shorter than T2 which is the 'count down' time from the negative reference voltage to zero.

TRIPLE SLOPE - INDIRECT INTEGRATING TYPE:

Is virtually identical to the dual slope type, with one significant addition: TWO time periods are integrated by the clock in addition to the second count down. The first is a 'rough' approximation, and the second a more 'detailed' approximation. This yields a more accurate digital output, BUT the price paid for it is 'TIME.' A good analogy of the triple slope A/D converter compared to the dual slope units would be: 'reading a slide rule through a magnifying glass.'

CHARGE BALANCING - INDIRECT INTEGRATING TYPE

Is also called the quantized feedback type. This type is really a 'voltage to frequency converter' with additional counting and timing circuitry. It also utilizes an operational (active) integrator, comparator, and negative reference voltage source. The output pulse rate (frequency) is proportional to input voltage and continues zapping out constant width pulses to the counter until the 'fixed' timer turns everything off. The counter's output forms the digital word proportional to the analog voltage input.

Since it too is an indirect integrating type, IF the timer is synchronized with an external noise frequency, near infinite rejection of the noise is possible.

A somewhat analagous gizmo could be built using a 55 cent LM555 multivibrator (square wave oscillator) with a VARICAP (variable capacitance diode) sensing the analog voltage input and varying the output frequency accordingly. All one need add is a simple TRS-80 assembly language frequency counter program that would 'count' the number of pulses (keep the frequency quite low) and they would then have their 'very own homebrew' system using the analog-voltage-to-frequency-to-digital-voltage technique that is sort-of-similar to this approach.

SERVO/COUNTER FEEDBACK TYPE:

Goes back a few years and is one of the least complicated varieties. It utilizes a counter that 'steps' up the input to a D/A converter one count a time and then compares the D/A output with the analog voltage input. When the comparison is zero, then the counter's value is translated into the digital word representing the voltage. Its primary disadvantages are that it is relatively slow and is easily fooled by noise.

TRACKING TYPE:

Is a variation of the 'servo/counter type' theme. An up/down counter is used so that the D/A converter's output may go in EITHER direction to NULL the analog input voltage. Its most important advantage is that it can follow a changing input voltage IF the change is small, which it can follow quite rapidly. Hence the name, TRACKING. If the change is large, it is rather slow and offers little advantage over its ancestor, the servo/counter type.

FLASH (PARALLEL) TYPE:

This is the 'fastest gun' in the A/D converter family. Like all good performers, its services to not come cheap. It is also occasionally called the 'simultaneous type' as in the really high-speed conversion versions, some as fast as 100 MHz

off-the-shelf today. A good example would be their use in multiple-independently-targetable re-entry vehicles. We do NOT want a SLOW A/D converter in a nuclear tipped missile that cannot differentiate between Boston and Buzzrakistan. Some 'flash' A/D converters are even approaching conversion speeds of 1000 MHz in a few developmental laboratories.

Other than guided missiles, flash type A/D converters are widely used in radar, video and wide-band microwave modulation applications where speeds in the 5 to 20+ MHz range are necessary. Here is how they work:

1. Let us assume we want to flash convert an analog voltage into an 8 bit word and that the analog voltage varies from zero to +5.12 volts dc. As such, every 20 millivolts = 1 bit as 256 times .020 = 5.12.

2. Now, if we stack up 255 voltage comparators in a vertical row with each one biased to turn 'on' at a voltage 20 millivolts higher than the one beneath it, and then connect all 255 comparator's inputs in parallel to the unknown analog voltage source, then ipso facto, all comparators' outputs below the unknown voltage would be 'on' and all those above it would be 'off.'

3. All that remains to be done is to feed these 255 outputs to a 255 to 8 bit binary decoder and SHAZAM, we have our 8 bit word representing the analog voltage input. The first step is called 'quantization' and the second step, not too suprisingly, 'decoding.'

There are many variations upon this theme, depending upon the application's speed requirement and probably more important, the size of your budget.

Is it possible to build a homebrew 'flash' converter? Sure it is IF you have unlimited patience and a modicum of skill using high speed gates, comparators, and multiplexers. If you are really ambitious and not afraid of difficult challenges, by all means give it a try.

First, read up on the subject and start with a 15 bit rather than 255 bit converter unless you have either IBM's or Bell Lab's facilities at your disposal. Don't expect to obtain speeds much beyond the lower video frequencies using hybrid techniques (mixing discrete and monolithic components) unless you have a laser available for trimming thin-film resistor networks which are the 'heart' of the comparator biasing scheme. Remember, physical spacing costs TIME.

Probably the best advice we can give would-be flash converter experimenters is to thoroughly 'search' the SURPLUS electronics houses who occasionally come up with a modest sized batch of units that some misguided 'bean counter' has foolishly sold by the pound. Really high-speed 'factory-new' units cost an arm, leg, and your car too.

SUCCESSIVE APPOXIMATION - FEEDBACK TYPE:

Here is the answer to the hacker's prayer: 'Lord, please give me an A/D converter that has relatively high-speed capability that I can afford.' It is a member of the feedback family which includes both the servo/counter and tracking types, BUT that is where the similarity ends as it is truly a 'clever' approach to having your cake and eating it too.

We saved the best all-around A/D conversion system for last, considering speed-versus-cost-versus-availability circa 1981/1982, since we had high hopes that you would at least 'read and be exposed' to the other options available to you in the preceeding pages. 'A little knowledge is NOT a dangerous thing,' IF used with the wisdom and judgement you now have to select the most cost-effective A/D converter for your specific application.

Figure 6-6 is a block diagram of a successive approximation A/D converter that is typical of the ones readily available today at relatively modest cost. A bit later we will discuss A/D single chip data acquisition SYSTEMS; i.e., most everything on one chip including multiplexer, address decoder & tri-state latch. In both cases we will utilize successive approximation A/D conversion, so best PAY ATTENTION, Gridley.

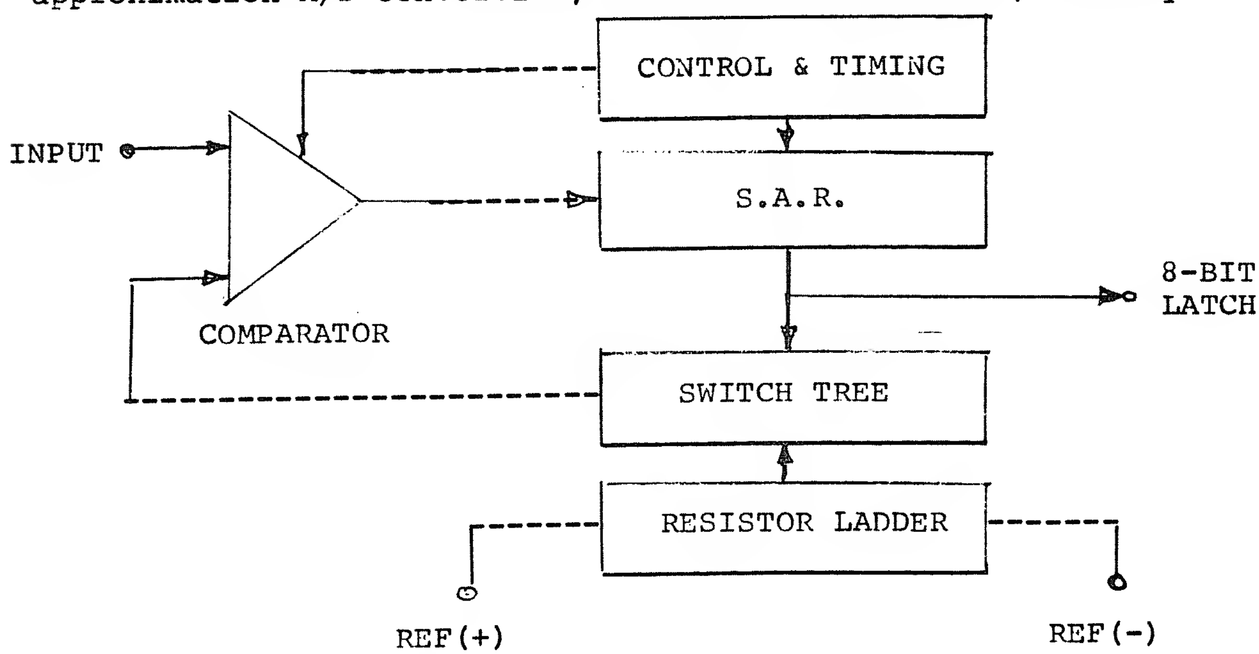


Figure 6-6

The control and timing block includes a D/A converter. S.A.R. is the successive approximation register and the switch tree is a bundle of solid state switches connecting the resistor ladder to the comparator. Here is how this clever design works for an 8 bit A/D conversion:

1. After "go," the MSB (= 10000000) of the 8 bit D/A converter is turned 'on' and one-half of the voltage across the

resistor ladder fed to the comparator and compared with the unknown analog voltage. IF the unkown voltage is equal to OR greater than the reference voltage THEN the MSB of the S.A.R. is set to a '1' and if NOT, then set to a zero.

2. With the first iteration completed, the second gets under way. The second MSB of the D/A converter (=01000000) is now turned 'on' and 1/4 of the voltage across the resistor ladder fed to the comparator by the switch tree. Again, if the unknown analog is equal to or greater than this new reference voltage THEN the next MSB of the S.A.R. is set to a '1' ELSE set to a zero.

3. For our 8 bit converter, this iteration continues to COMPARE 1/8th, 1/16th, 1/32nd, 1/64th, 1/128th, and 1/256th of the reference voltage and set the S.A.R. as appropriate. When it is all finished SUCCESSIVELY APPROXIMATING the unknown analog voltage, a total of 8 iterations and compares have been made. IF we required 12 bit output, then a total of 12 iterations/comparisons would have to be made.

4. After the 8th comparison has been made and stored in the S.A.R., some of the successive approximation A/D's output the 8 bit word from the S.A.R. to an 8 bit tri-state latch/buffer and then output an EOC (end-of-conversion) flag/signal announcing, "I'm all done. What next?"

Compared to the counter type A/D converter which compares all 256 bits to complete a conversion, the successive approximation variety is a supersonic-whiz-bang with only 8. Many are able to complete A/D conversion of the unknown voltage to an 8 bit word in considerably less than a microsecond with the average moderately priced units performing a conversion every 100 microseconds = 10,000 per second with an accuracy of plus or minus less than 1/2 least significant bit. A few years ago, the designer of an A/D converter with this price/performance ratio would have won the Nobel prize.

NATIONAL SEMICONDUCTOR ADC0808 A/D DATA ACQUISITION SYSTEM:

Here is an example of superlative (super-great-fabulous) A/D successive approximation converter design at its cost-effective best, PLUS also ON THE SAME CHIP:

1. 8 channel multiplexer analog switches that allow 8 separate and independent unknown analog voltage inputs.

2. 3 bit binary address decoder and latch for selecting any ONE of the above inputs.

3. 8 bit tri-state output buffer/latch for converted word.

4. EOC (end of conversion) flag/signal output.

In addition to all these goodies on a single 28 pin chip which

is selling in the $30 - $50 ballpark (fall 1980), National Semiconductor also manufactures the ADC0816 which is identical EXCEPT that it offers 16 multiplexed/switched input channels on a 40 pin DIP chip. Price is approximately the same as the ADC0808. This should make our friends in the chemical processing and/or nuclear industries happy who have to monitor umpteen different unknown analog inputs ALMOST simultaneously.

Figure 6-7 is a block diagram of the ADC0808 courtesy of Natl. Semiconductor.

Block Diagram

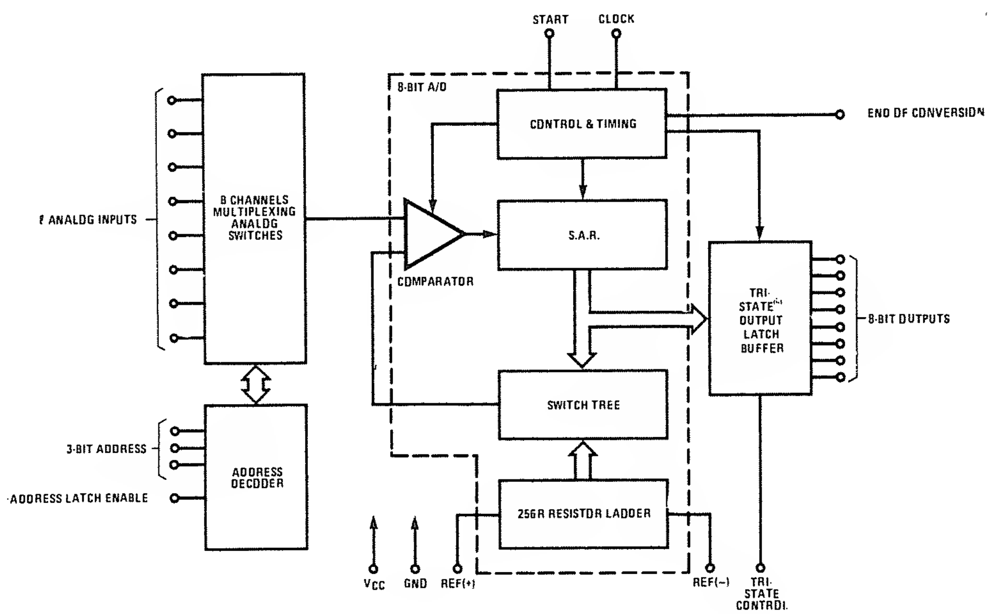
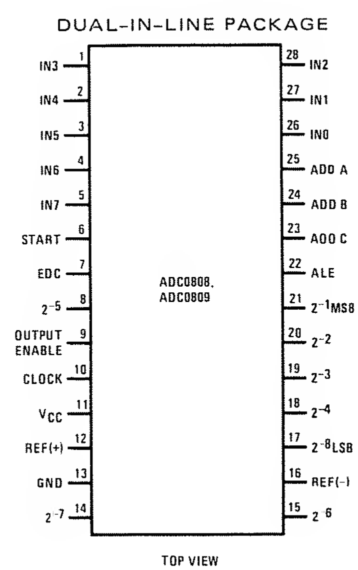


Figure 6-7

Is of course identical to Figure 6-6 on the last page, except for the extremely convenient and useful additions discussed above. Figure 6-8 is the pin out diagram for the ADC0808 and ADC0809. The 09 version is identical to the 08 except for temperature range AND unadjusted error accuracy by 1/2 LSB. Rumor has it that the 09 version does not exist as ALL the production units meet 08 accuracy specifications.

Figure 6-8 →



- THE ALPHA PRODUCTS 'ANALOG 80' -

8 CHANNEL A/D DATA ACQUISITION SYSTEM FOR THE TRS-80:

Is an unusally fascinating, brilliantly engineered, and well manufactured accessory for the TRS-80. If THAT sounds like we like it and are impressed with it, WE ARE. It is built by the Alpha Product Company, 85-71 79th Street, Woodhaven, NY 11421 and lists for $140. including power supply, as of fall 1980. Figure 6-9 below illustrates the 3 1/2" wide by 6" long by 1.5" high case and 40 conductor cable with 40 pin connector for attachment to either the TRS-80 keyboard or expansion interface's screen printer port.

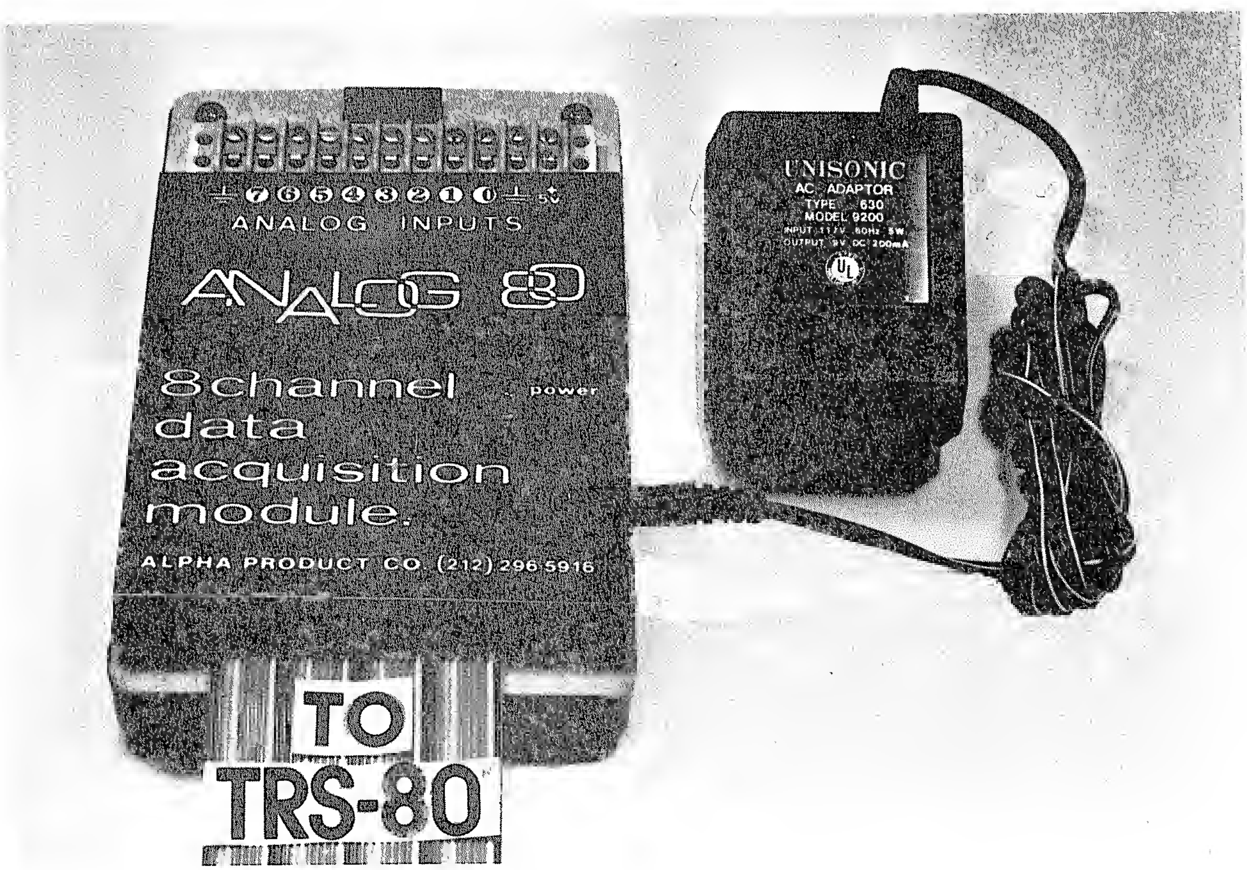


Figure 6-9

Dr. John M. Monin, the designer of the Analog 80 and General Manager of Alpha Product Company asked us to inform our readers that they may obtain a 10% discount on this unit IF they will mention the "Disassembled Handbook" on their order. Enough crass commercials for what we think is a fine product. Now, let's dig into it and see what goodies lurk beneath its cover.

The top of the case contains the terminal strip with inputs for 8 analog channels, plus a ground terminal at each end. A red LED in the right center of the case reminds us whether power is on or off. EACH individual analog input is via a 22,000 ohm 1/4 watt resistor that feeds the ADC0808 input pins 27, 26, 25, 1, 2, 3, 4, and 5 for channels zero through 7. Also, EACH of the 22K resistors is bypassed for extraneous unwanted ac pickup to ground with .047 microfarad bypass capacitors on the ADC0808 side of the input resistors.

A REALLY CLEVER INNOVATION:

After removing the Analog 80's cover you will note the individual 22K isolating resistors and bypass capacitors for each channel just below the input terminal strip. WITHOUT any modification, the Analog 80 is shipped with each input channel setup to digitally output 255 decimal (using the TRS-80 INP function) for a full scale reading of +5.12 volts dc and zero for zero volts dc input thus giving a spread of 20 millivolts per digit; i.e., again, 256 X .020 = 5.12 volts dc.

Now, NOT everyone is necessarily making RATIOMETRIC readings of voltage sources easily convertible to 5.12 volts dc full scale. One might wish to utilize an input source that is doing other good things, at say 10.24 volts dc full scale, or what-have-you. Dr. Morin thoughtfully included holes in the printed circuit board adjacent to EACH input 22K resistor so that the user could EASILY solder in 1/4 watt resistors to form a voltage divider for EACH channel. They go from the inputs' 22K resistor junction with the .047 mfd bypass capacitor to ground so that IF you wished 10.24 volts dc full scale all that is necessary is to solder in a 22K ohm 1/4 watt resistor for this particular channel in the holes ALREADY provided for it on the printed circuit board. IF you wanted 51.2 volts full scale, then solder in a 'trimmed' (with a file) 2.2K ohm 1/4 watt resistor for that particular channel, etc.

Though you could carry this on to the 'absurd' level of say a 220 ohm resistor to read 512 volts dc, it is suggested that you hold your input levels to +100 volts dc or lower to protect the rather delicate TTL gates involved from ground loops and other weird beasties that like to 'kill' low power Schottky gates just for the fun of it. Though the price of the ADC0808 has dropped to the $30-$50 range, blowing one of them may spoil your whole day, or weekend. Best 'play it' conservative by not intentionally courting disaster.

ONE NOTEWORTHY ITEM OF CONSIDERABLE IMPORTANCE:

The input impedance of the ADC0808 multiplexer is extremely high which is a vote in its favor as it will have virtually NO effect on your unknown voltage source. Since the 22K ohm input resistor for each channel is floating, assuming you have NOT installed a voltage divider in the PCB board holes provided for one, stray-extraneous voltage pickup will cause MEANING-LESS readings on those channels that are NOT connected to anything. All that is necessary to eliminate this pickup on unused channels is to tie a 1 megohm 1/4 watt resistor to ground from each unused channel's input. Conversely, the 1 megohm resistors may be installed in the holes already drilled through the printed circuit board on the ADC0808 side of the 22K ohm input resistors. The induced error caused by the 1 megohm resistor acting as a voltage divider is only 2 percent, so it may be left in place IF desired. Just remember to remove it later if maximum readout accuracy is necessary.

DIGGING A BIT DEEPER INTO THE ANALOG 80:

You will recall from the last Chapter, that the VAR/80 used only a single address bus bit to decode the PORT zero instruction for the sake of economy AND to avoid over loading the address bus on early models. Well now, the Analog 80 offers a considerably more sophisticated circuit that allows the user to choose between PORTs 0, 1, 2, 3, 4, 5, 6, or 7 by utilizing a manually programmed 16 pin DIP socket and MOST of a 74LS42 BCD to decimal decoder to 'ding-a-ling' whichever PORT's bell you wish to ring in your particular application.

The 16 pin DIP socket for programming the PORT desired is loacated on the left mid-section of the printed circuit board, just beneath the cover. A simple #22 OR #24 wire jumper is installed as shown below, for the PORT desired:

TOP VIEW OF 16 PIN DIP SOCKET

PORT	pin#	U	pin#
0	1	----------------	16
1	2	----------------	15
2	3	----------------	14
3	4	----------------	13
4	5	----------------	12
5	6	----------------	11
6	7	----------------	10
7	8	----------------	9

The jumper is supplied, and initially connected betweem pins #1 and #16 for PORT zero. Additional jumpers ARE NOT REQUIRED. You need only MOVE this jumper to allow the 74LS42 to decode the PORT you wish to use. Address bus lines A0, A1, and A2 are decoded by the 74LS42 to determine the PORT desired. As such, ANY port number that is 8, 16, 24, etc. PLUS the port number will also be decoded; i.e., 7 + 248 = 255; be cautious.

EXPANDABUS OPTION:

Should you require more than one Analog 80's 8 channels, then we suggest you try one of the Alpha Product's EXPANDABUS accessory expansion cables that may be plugged into either the keyboard or expansion interface to allow up to 4 Analog 80's or other accessories to be interconnected to the TRS-80's 40 pin bus at one time. We have not tried it, but Dr. Monin assures us that it works, "fine business."

PROGRAMMING WITH THE ANALOG 80:

Is about as straightforward as could be desired with an 8 channel data acquisition system. Let's assume that the jumper on the port selection DIP socket is in place between terminals 1 and 16, thus selecting PORT zero.

1. First we must tell the Analog 80 WHICH of the eight channels, 0 through 7, which we wish to access. This is done with the OUT0,X statement in BASIC, where 'X' is the 0 through 7 channel we wish to read; i.e., OUT0,5 selects channel 5. If we were doing this in assembly language we would first LD A,5 and then OUT (0),A to accomplish the same thing.

2. The OUT statement also serves another function through a very ingenious ploy by Dr. Monin. It initiates a NEW conversion cycle for the ADC0808. Until a NEW out statement is received, the ADC0808's tri-state output latch/buffer holds the value of the LAST conversion.

3. After OUT0,5 the value of the NEWEST conversion is now available to the data bus and may be read out by PRINTINP(0) or A=INP(0):PRINTA in BASIC. If we were NOT using a resistor voltage divider on the ANALOG 80's channel 5 we could just as well print out the actual dc voltage reading by multiplying A times 20 millivolts and use PRINTINP(0)*.02 or A=INP(0)*.02: PRINTA, if that suits your fancy. In assembly language we would simply use the IN A,(0) instruction, LD A into the ACCUM's integer LSB memory location at 4121H, make sure that the number type flag at 40AFH = 2 = integer, and then display its value on video with a CALL 0FBDH and CALL 28A7H.

A SIMPLE BASIC PROGRAM TO READOUT ALL ANALOG 80 CHANNELS:

```
10 CLS:X=0:B=INP(0)
15 OUT0,X
20 A=INP(0)
25 PRINT"CHANNEL";X;"= ";A*.02;"VOLTS D.C.
30 X=X+1
35 IFX=8THENX=0:GOTO45
40 GOTO15
45 A=INP(0):IFA<>BGOTO10ELSE45
```

THAT BASIC PROGRAM ON THE LAST PAGE WON'T WORK ! ! ! YOU JUST TOLD US THAT THE ANALOG 80 REQUIRES AN 'OUT' TO RESET ? ? ? ?

Thank you, Gridley. You are really sharp today and it is most certainly a fair and logical question. I am glad you asked it since it leads us into another feature of the Analog 80 that is extremely useful, the EXTERNAL STROBE.

There are a number of occasions when we only wish to renew/ update the readings when some external device 'ASKS' for another reading. One way to accomplish this with the Analog 80 is to furnish pin 8 on the external 16 pin DIP connector at the top of the case with a +5 volt dc pulse. This pin is normally held LOW via a 1000 ohm resistor to ground. Whenever a brief +5 volts dc pulse, such as you would obtain by connecting a NORMALLY OPEN pushbutton switch to pin 8 and a +5 VDC source and pushing it briefly, then the Analog 80 would take ANOTHER reading and update its tri-state output buffer/latch.

The mini-program on the last page is setup to take a look at all 8 channels IF the external strobe asks for one and IF channel zero, the one we presume you are interested in, has changed from the previous reading. By all means change it to suit your particular application, as it is only an example. The illustration below shows the pinout connections to the 16 pin DIP socket on the top of the Analog 80 case.

	pin#	U	pin#	
8-10 VDC unreg.	1		16	channel 0
ground	2		15	channel 1
ground	3		14	channel 2
no connection	4		13	channel 3
no connection	5		12	channel 4
ground	6		11	channel 5
+5.12 VDC reg.	7		10	channel 6
external strobe	8		9	channel 7

Assuming you had a 16 conductor jumper terminated with a 16 pin DIP plug, this socket is a real convenience for inter-connecting the Analog 80 to most any form of external inter-face.

Yet another option, and probably one of the most useful, would be to use an external interrupt to zap whatever program was in progress off to an assembly language subroutine that would take a reading of all eight Analog 80 input channels whenever called, and then return to the normal program. Using the interrupt mode one scheme we covered in Chapter 4, with the Analog 80 connected directly to the keyboard's 40 pin connector, let's have a go at writing a really simple assembly language subroutine that will DO just that when called.

EVERY TIME YOU SAY 'SIMPLE' I FASTEN MY SEATBELT ! ! !

Relax, Gridley. There's no need for you to become gun-shy.

HYBRID BASIC & ASSEMBLY LANGUAGE PROGRAMS FOR THE ANALOG 80:

Here are two relatively straightforward programs to illustrate ONE approach to using the interrupt Mode one to initiate a reading from all 8 channels of the Analog 80. We will use a BASIC program to do something useful. The simplest thing we could think of was a plain 'ole counter, but you may substitute ANYTHING you wish from fast Fourier transforms to calculating the distance to the moon in furlongs. The BASIC program clucks along merrily counting away until an interrupt is received. Then all sorts of good things happen.

Upon receipt of the interrupt, you can easily do this by momentarily grounding pin 21 of the keyboard's 40 pin connector with a normally open pushbutton switch, the Z-80 automatically issues an RST 38H which causes ROM to JumP to 16402 in MEM where we cleverly put a JumP to 32000 in MEM where our assembly language program is located.

This program then rings the Analog 80's bell by initiating a conversion for channel zero, followed by a 1/6th second time delay (1/6000th probably ok) to accomplish the conversion. It then stores channel zero's converted digital value in MEM location 32100 (easy to remember). This is repeated for channels 1 through 7 with their converted values stored in MEM locations 32101 to 32107. The program then re-enables the interrupt flipflop IFF1 and RETurns to wherever it was interrupted in the BASIC program. There is nothing even remotely sophisticated about these two programs, but they DO (as in Fortran) show how extremely easy it is to:

1. Use the Analog 80 with assembler.

2. Interleave a BASIC program and assembly language program.

3. Use the interrupt Mode 1 to initiate a reading of all eight channels fed to the Analog 80.

TO AVOID BOLIXING UP THE EXPANSION INTERFACE, THESE PROGRAMS ARE RUN WITH THE ANALOG 80 CONNECTED 'DIRECTLY' TO THE TRS-80.

```
10 'BASIC PART HYBRID INTERRUPT ANALOG 80 PROGRAM - READ
20 '
30 POKE16402,195:POKE16403,0:POKE16404,125:DEFINTA-Z
40 CLS:FORA=32100TO32107:POKEA,0:NEXT
50 B=32100:C=PEEK(32100)
60 FORD=0TO7:PRINT"CHANNEL";D;"VOLTS D.C. =";PEEK(B)*.020
70 B=B+1:NEXT:PRINT
80 PRINT"I AM DOING SOMETHING ELSE TILL AN INTERRUPT REQUEST HAS"
90 PRINT"BEEN RECEIVED 'AND' CHANNEL ZERO HAS CHANGED ITS VALUE."
100 F=F+1:E=PEEK(32100):IFC<>ETHENCLS:GOTO50
110 PRINT@794,F:GOTO100
```

```
00100 ;ASSEMBLY LANGUAGE PART OF HYBRID ANALOG 80 PGM - RD1/RD2
00110 ;
00120             ORG     32000         ;START THE PROGRAM HERE
00130             EX      AF,AF'        ;SWAP ALTERNATE REGISTERS
00140             EXX                   ; "    "         "
00150 ZERO        LD      A,0           ;CHANNEL ZERO
00160             CALL    READ          ;GOTO READ
00170             LD      (32100),A     ;STASH IT AWAY AT 32100
00180 ONE         LD      A,1           ;CHANNEL ONE
00190             CALL    READ          ;GOTO READ
00200             LD      (32101),A     ;STASH IT AWAY AT 32101
00210 TWO         LD      A,2           ;CHANNEL TWO
00220             CALL    READ          ;GOTO READ
00230             LD      (32102),A     ;STASH IT AWAY AT 32102
00240 THREE       LD      A,3           ;CHANNEL THREE
00250             CALL    READ          ;GOTO READ
00260             LD      (32103),A     ;STASH IT AWAY AT 32103
00270 FOUR        LD      A,4           ;CHANNEL FOUR
00280             CALL    READ          ;GOTO READ
00290             LD      (32104),A     ;STASH IT AWAY AT 32104
00300 FIVE        LD      A,5           ;CHANNEL 5
00310             CALL    READ          ;GOTO READ
00320             LD      (32105),A     ;STASH IT AWAY AT 32105
00330 SIX         LD      A,6           ;CHANNEL 6
00340             CALL    READ          ;GOTO READ
00350             LD      (32106),A     ;STASH IT AWAY AT 32106
00360 SEVEN       LD      A,7           ;CHANNEL 7
00370             CALL    READ          ;GOTO READ
00380             LD      (32107),A     ;STASH IT AWAY AT 32107
00390             EX      AF,AF'        ;RESTORE ORIGINAL REGS.
00400             EXX                   ; "      "        "
00410             EI                    ;RE-ENABLE INTERRUPTS
00420             RET                   ;POP STACK RETN TO BASIC
00430             ORG     32200         ;MOVE PROGRAM TO 32200
00440             IM      1             ;SET INTERRUPT MODE ONE
00450             EI                    ;SET IFF1 "INT ENABLED'
00460             JP      114           ;RETURN TO BASIC 'READY'
00470 READ        OUT     (0),A         ;READ CHANNEL IN 'A'
00480             LD      BC,10000      ;1/6TH SECOND DELAY
00490             CALL    060H          ;ROM DELAY SUBROUTINE
00500             IN      A,(0)         ;PORT 0 VALUE TO 'A'
00510             RET                   ;RETURN LINE AFTER CALL
00520             END     32200         ;INITIALIZE AT 32200
```

Source Code

```
7D00             00120            ORG     32000
7D00 08          00130            EX      AF,AF'
7D01 D9          00140            EXX
7D02 3E00        00150 ZERO       LD      A,0
7D04 CDCE7D      00160            CALL    READ
7D07 32647D      00170            LD      (32100),A
7D0A 3E01        00180 ONE        LD      A,1
7D0C CDCE7D      00190            CALL    READ
7D0F 32657D      00200            LD      (32101),A
7D12 3E02        00210 TWO        LD      A,2
7D14 CDCE7D      00220            CALL    READ
7D17 32667D      00230            LD      (32102),A
7D1A 3E03        00240 THREE      LD      A,3
7D1C CDCE7D      00250            CALL    READ
7D1F 32677D      00260            LD      (32103),A
7D22 3E04        00270 FOUR       LD      A,4
7D24 CDCE7D      00280            CALL    READ
7D27 32687D      00290            LD      (32104),A
7D2A 3E05        00300 FIVE       LD      A,5
7D2C CDCE7D      00310            CALL    READ
7D2F 32697D      00320            LD      (32105),A
7D32 3E06        00330 SIX        LD      A,6
7D34 CDCE7D      00340            CALL    READ
7D37 326A7D      00350            LD      (32106),A
7D3A 3E07        00360 SEVEN      LD      A,7
7D3C CDCE7D      00370            CALL    READ
7D3F 326B7D      00380            LD      (32107),A
7D42 08          00390            EX      AF,AF'
7D43 D9          00400            EXX
7D44 FB          00410            EI
7D45 C9          00420            RET
7DC8             00430            ORG     32200
7DC8 ED56        00440            IM      1
7DCA FB          00450            EI
7DCB C37200      00460            JP      114
7DCE D300        00470 READ       OUT     (0),A
7DD0 011027      00480            LD      BC,10000
7DD3 CD6000      00490            CALL    060H
7DD6 DB00        00500            IN      A,(0)
7DD8 C9          00510            RET
7DC8             00520            END     32200
00000 TOTAL ERRORS

READ    7DCE 00470    00160 00190 00220 00250 00280 00310 00340
                      00370
```

Object Code

NOTE: Labels zero through seven are reminders and serve NO useful purpose.

HOW DOES IT WORK ? ? ?

Glad you asked, Girdley. Always the right question at the right time. It works quite well, thank you.

Seriously, let's go back a few pages and look at the BASIC part of these hybrid programs first.

Line 30: pokes the JumP instruction into 16402, 32000 into 16403 and 16404, plus defines integers for speed.

Line 40: CLS and then zeroes out our channels 0 through 7 MEM stash locations.

Line 50: sets variable B to 32100, the beginning stash MEM loacation and variable C to channel zero's initial value so as to see if it has changed later.

Line 60: prints out the channel number and voltage of each channel consecutively.

Line 70: increments the MEM location by +1, has the NEXT for line 60's FOR and prints a line space.

Lines 80 & 90: print out the TRS-80's message.

Line 100: variable F is our counter to illustrate that the BASIC part of the program is doing 'something' between interrupts. Variable E is = to the most recent value of channel zero, and is compared with C to see if there has been a change. IF so, then CLS and GOTO 50 to print out the newly converted values. Otherwise drop down to line 110.

Line 110: Print out counter F's value and go back to line 100.

Now let's look at our assembly language source code program in detail that begins at 32000 decimal in MEM. With a few minor exceptions it is quite similar to our earlier BASIC program on page 6-21 that read out all of the Analog 80's eight channels.

The ZERO through EIGHT labels in the left column are completely UNNECESSARY and ONLY included as a reminder for typists with poor memories, like the author.

YOU ADMIT THAT YOUR MEMORY IS FAILING ? ? ?

Sure I do, Gridley. Remember to have yours checked in another 50 years. Maybe, you might have it checked tomorrow?

Lines 140 through 380 quite simply ask the Analog 80 to take a reading for each channel in succession and store it in MEM locations 32100 through 32107. The delay in lines 480 and 490 is to give the ADC0808 time to make the conversion. As we previously mentioned, a millisecond delay is probably enough.

Lines 390-400 switch the original registers AF, BC, DE, and HL back into service with whatever values they contained BEFORE the interrupt. Remember, the IY register is NEVER used by level 2 ROM, and the IX register only very seldom as the Level II BASIC written by Microsoft's Paul Allen and Bill Gates is the 'son of' an earlier 8080 BASIC which has neither IX or IY. Saving the IX register in the stack is easy enough to accomplish if you wish. Just PUSH IX in line 145 and POP it out again in line 375.

You will recall that a maskable interrupt 'turns off' interrupt flip-flop IFF1. Line 410 re-enables IFF1. Line 420 POPs the stack with the exact address of wherever your BASIC program WAS when the interrupt occurred and sends it back to continue the BASIC program as IF the interrupt never happened. If you choose to use the stack in interrupt subroutines, you MUST pay attention and keep your PUSHes and POPs EXACTLY even so that this RETurn address that the interrupt automatically stored in the stack is in the correct stack location when it is time to go back to the interrupted program.

Line 430 instructs the editor/assembler to locate further instructions at 32200 on up. Why? Two reasons: first, we want to be able to intitialize the program 'up' out of the way of the interrupt subroutine, and second, MEM location 32200 is easier to remember than 32208. Line 440 tells our Z-80 that we will being using interrupt mode one, henceforth. It could just have well been IM zero or IM 2, except they will not work with the TRS-80 UNLESS we choose to butcher up some rather complicated circuitry. Line 450 enables interrupts as Level II WITHOUT the expansion interface attached, powers up with ALL maskable interrupts DISABLED. Line 460 sends us back to BASIC with a 'READY' after intialization. DO NOT use JP 1A19H as some books suggest as it will often foul-up any BASIC resident program, thus forcing you to reload it from cassette.

RUNNING THE HYBRID ANALOG 80 PROGRAM:

We assume that you have both of the programs on cassette and the Analog 80 plugged into the TRS-80 keyboard so as NOT to bolix up the interrupt mode one circuitry in the expansion interface that is used by the floppy disks and sort-of real time clock. One option that is very convenient IF you plan to use INTERRUPTS frequently, (unplugging the keyboard connector is NO fun at all), is to use the Alpha Product's EXPANDABUS adaptor BETWEEN the keyboard and the expansion interface buffered 40 conductor cable. This allows you to plug the Analog 80 into one of the four male 40 pin connectors on the EXPANDABUS adapter. To use an interrupt driven program, it is ONLY necessary to turn 'OFF' the expansion interface. No mucking about changing connectors and cables is required.

You may load either the BASIC or object code first. It makes no nevermind. Just REMEMBER that after loading the SYSTEM tape, type in /32200 then ENTER to intialize IM 1 and to EI.

Some TRS-80s have a proclivity for 'forgetting' the entry point (address) of a given system program which should be stored at MEM locations 40DFH and 40E0H. If you use the /32200 then ENTER after loading the SYSTEM program, you will avoid this potential problem.

PROGRAM VIDEO DISPLAY:

Will look like the print out below when first run. REMEMBER to use 1 megohm resistors to ground, on unused Analog 80 input channels. In this program we had channel zero AT 5.12 VDC.

```
CHANNEL 0 VOLTS D.C. = 0
CHANNEL 1 VOLTS D.C. = 0
CHANNEL 2 VOLTS D.C. = 0
CHANNEL 3 VOLTS D.C. = 0
CHANNEL 4 VOLTS D.C. = 0
CHANNEL 5 VOLTS D.C. = 0
CHANNEL 6 VOLTS D.C. = 0
CHANNEL 7 VOLTS D.C. = 0

I AM DOING SOMETHING ELSE TILL AN INTERRUPT REQUEST HAS
BEEN RECEIVED 'AND' CHANNEL ZERO HAS CHANGED ITS VALUE.

      counter -----> 999 <----- BREAK at this value
```

Is anything wrong with the circuit or program? No, it takes one interrupt to initiate the ADC0808 conversion process. Now press your interrupt pushbutton switch. Video output is below.

```
CHANNEL 0 VOLTS D.C. = 5.1
CHANNEL 1 VOLTS D.C. = 0
CHANNEL 2 VOLTS D.C. = 0
CHANNEL 3 VOLTS D.C. = 0
CHANNEL 4 VOLTS D.C. = 0
CHANNEL 5 VOLTS D.C. = 0
CHANNEL 6 VOLTS D.C. = 0
CHANNEL 7 VOLTS D.C. = 0

I AM DOING SOMETHING ELSE TILL AN INTERRUPT REQUEST HAS
BEEN RECEIVED 'AND' CHANNEL ZERO HAS CHANGED ITS VALUE.

                           1345
```

Channel zero reads out 5.1 volts dc instead of 5.12 VDC. Is anything wrong? No, it is working perfectly. The 5.10 reading is the maximum full scale value it is capable of converting as it must count zero volts dc as the lowest possible reading leaving 255 (out of a possible 256) for higher values. Simple arithmetic has 255 times .02 = 5.1, so it is doing exactly what it is supposed to do.

A/D converters are extremely useful gizmos IF you need to interface to the outside real world. Now, have a go at the questions for this chapter and THEN we'll tackle D/A systems.

- CHAPTER 7 -

AN INTRODUCTION TO DIGITAL TO ANALOG CONVERTERS

INTRODUCTION:

"The world is full of many wonderful things," the poet said. "Most of these things are analog," the digital computer said.

Well now, just how do we control that largely analog real world out there with our TRS-80, Gridley?

I'VE GOT IT! JUST HOOKUP UP OUR VAR/80 OR INTERFACER 2 TO SOME ADDITIONAL DEMULTIPLEXERS FOR DECODING 'ALL' 256 BITS FROM A TRS-80 PORT. THEN WE'LL HAVE EACH OUTPUT DRIVE A BUFFER THAT CAN ACTUATE A RELAY ATTACHED TO A RESISTOR LADDER. THAT WAY WE CAN OUTPUT ANY ANALOG VOLTAGE FROM ZERO TO MAXIUM IN 255 STEPS ! ! ! IT WOULD WORK, WOULDN'T IT ? ? ?

Sure it would, Gridley. There is only one minor problem. How would we be able to fit ourselves into the computer room with 255 relays and all sorts of buffers and demultiplexers taking up all the space?

S-I-L-E-N-C-E

Don't feel bad, Gridley. It was a good idea IF space and money were of no consequence. However, these is a much easier and less expensive way to do it. Care to guess, Gridley?

A DIGITAL TO ANALOG CONVERTER, CAUSE THAT'S WHAT THIS CHAPTER IS ALL ABOUT ! ! !

Right Gridley, you are sharp as ever today. You took the words right out of my mouth. Your idea was not ALL bad, Gridley. Actually, a bit later we will need a latching output for the PORT we will be using and both the VAR/80 and Interfacer 2 provide us with this very necessary capability. We COULD build our own latching PORT interface out of a few chips, but since we already have a VAR/80 attached and working, we will skip duplicating that subject since it was covered a few Chapters back.

First off, we will build our own D/A converter using the TRS-80 to do most of the conversion work. It will NOT be a very sophisticated one, but it will work. Secondly, we will take a quick run through the more popular conversion schemes in use today. Then, we will construct two D/A converters using inexpensive National Semiconductor 8 bit D/A's in the under $4. price class. Both of these units will have remarkably good performance/price ratios. A few years ago, similar performance would have cost us over 100 times as much. Now let's discover how very simple D/A converters really are.

THERE'S THAT 'SIMPLE' AGAIN. I'M GOING TO FASTEN MY SEAT BELT!

BUILDING A HOMEBREW D/A CONVERTER:

You will recall that the time constant of a resistor/capacitor combination, the time required for the capacitor to charge to 63.2 percent of the value of the source voltage THROUGH the resistor, was equal to the resistance in megohms times the capacitance in microfarads when time was in seconds. Graphing voltage versus time gave us a natural/Naperian logarithmic curve. Natural logarithms are what our TRS-80 likes best, so it was a 'snap' in the Chapter on A/D converters for our computer to calculate the digital value of the analog input to our converter. Well, 'tit for tat who ate the bat on the cover of Harv Pennington's book,' my cat said. 'Tit for tat,' indeed. Of course if we can make a homebrew A/D converter using the 'ole RC time constant equation:

$$e = \left(E \quad 1 - \varepsilon\right)^{-t/RC}$$

We can of course make a homebrew D/A converter using the identical equation. It should be NO great trick to work it backwards and have our TRS-80 output an analog voltage equal to whatever digital voltage we specify in our program. The values in the above equation were:

e = voltage across capacitor
E = source voltage
t = seconds
R = megohms resistance
C = microfarads capacitance
ε = base of natural logarithm

The ONLY components/gizmos we will need for this interesting experiment are a voltage source of let's say 51.2 volts dc, a 1 watt 20,000 ohm resistor, a 50 microfarad 450 working volt dc capacitor, and a vacuum tube voltmeter to see if our D/A converter is really working.

WHY A 20K RESISTOR AND 50 MFD CAPACITOR ? ? ?

Because 20K ohms in megohms = .02 X 50 = 1, and that's about the highest mathematics we will ask you to perform, Gridley. Seriously, $-t/RC$ is easier to work with when RC =1.

WHY A 450 WORKING VOLT CAP. WHEN WE'RE ONLY USING 51 VOLTS ? ?

You are just FULL of good questions today, Gridley. ALL capacitors have SOME problems and electrolytics are about the worst we could use. Their major good point is that they are CHEAP. Using a 450 WVDC electrolytic capacitor at 50 volts won't be TOO bad. Ideally, we would use a capacitor with zero dielectric absorption and infinite insulation resistance, but sad to say, such a device does not exist. Some types of capacitors are better than others regarding these two variables. Dielectric absorption is sometimes called 'voltage memory' and is caused by the dielectric's inablity to polarize

instantaneously. Unfortunately, many dielectric materials exhibit this 'voltage memory' and some are better than (not as bad as) others. The dielectric's molecular dipoles take 'time' to align themselves in an electric field. The was one of the major errors introduced by using an electrolytic in our home-brew A/D converter, in an earlier Chapter. If we were to rank the various types of capacitors available, in order of desirability along with their dielectric absorption we have:

RANK	TYPE	DIELECTRIC ABSORPTION
1.	teflon	.01%
2.	metalized teflon	.02%
3.	polystyrene	.02%
4.	polypropylene	.03%
5.	polycarbonate	.05%
6.	paper	fair
7.	ceramic	poor
8.	electrolytic	awful

Not too surprisingly, the cost per unit of capacitance is ALMOST inversely proportional to the ranking in desirability.

Actually, the picture is not as 'awful' as it appears for our first experiment in this Chapter. We are not particularly interested in how 'fast' our D/A converter works, so we can trade-off a chunk of accuracy for 'time' AND 'rig' (as in cheat) our BASIC program to correct for some of the dielectric error inherent in our electrolytic's construction.

Figures 7-1 and 7-2 are our old friends, the exponential curves illustrating charge and discharge voltage across a capacitor C with respect to RC/time, which in our case = 1 second per RC. Both drawings are courtesy of ARRL.

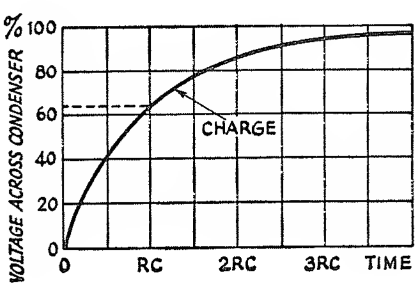


Figure 7-1

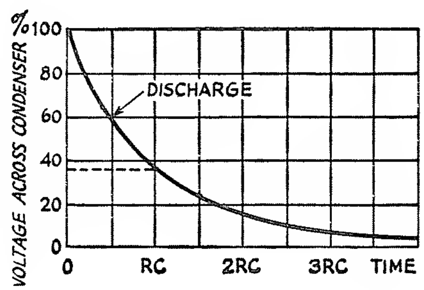


Figure 7-2

HOW WE GONNA MAKE IT WORK ? ? ?

Verrry easily, Gridley. We will write a program that asks us to INPUT whatever analog dc voltage we wish to output FROM our D/A converter. We KNOW that if we write a BASIC program that has a 2 line number loop with an IF-THEN stuck in, that it will make about 80 loops per second.

WHAT'S A 2 LINE NUMBER LOOP WITH AN IF-THEN STUCK IN ? ? ?

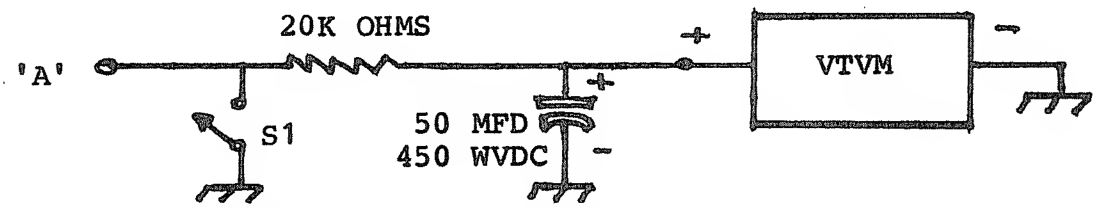
Try these two lines on for size, Gridley. Do they fit?

```
50 X=X+1:IFX=YTHENY=0:GOTO70
60 GOTO50
```

WHY DIDN'T YOU USE 'ELSE' AND SAVE A LINE ? ? ?

So as NOT to confuse the issue with an AND, Gridley. Kindly allow us to continue.

Here is the circuit we will use to charge up capacitor 'C' through resistor 'R' for the time period in seconds = T/80.



POINT 'A' TO VAR/80 DBO ZERO CENTER TERMINAL

Figure 7-3

The VTVM is our vacuum tube voltmeter we will use to measure the voltage across capacitor 'C.' The reason we use a VTVM is NOT load down our charged capacitor TOO quickly and thusly obtain an erroneous reading. MOST multimeters ONLY have an input resistance of 20,000 ohms per volt, whereas most VTVMs have an input resistance of 1 to 2 megohms (million ohms) or higher. The resistance of our voltage measuring device is obviously DRAINING OFF the current and thus reducing the voltage we wish to measure. The higher resistance of your voltmeter, the better.

HOW ABOUT A LEYDEN JAR WITH ALUMINUM FOIL SPREADERS ? ? ?

If you can read tea leaves, you can read a Leyden jar as a voltmeter. Go ahead and use one. Good luck, Gridley.

HOW ABOUT A 'SAMPLE AND HOLD' TO FEED OUR VOLTMETER ? ? ?

Gridley, you are getting too smart for your britches in this Chapter. I just heard the RECESS bell ring. Best NOT miss the freebie graham crackers and carton of milk. So long, Gridley.

NOW, we can do some serious work.

Point 'A' in Figure 7-3's schematic is connected to the the 'C' center terminal of our VAR/80 DBO zero relay contact and point 'B' to our vacuum tube voltmeter's input for a reading.

Switch S1 may be a normally open pushbutton switch, Radio Shack #275-1547, or any variety you wish to 'zero' out the voltage across capacitor 'C.' It is recommended that you NOT use a toggle switch since you may blow your power supply and ruin the VAR/80's DBO zero relay contacts IF you forget and leave it closed when running the program. Yet another option is to have the VAR/80'S DBO 1 normally open relay contacts replace switch S1 which then allows you to 'zero' out the voltage across capacitor 'C' under program control.

Remember, the voltage 'memory' problem of an electrolytic capacitor's dielectric is somewhat less than perfect. Try this little experiment:

1. Charge up 'C' to 100 volts dc. Open the charging line.

2. Press switch S1 until your VTVM reads reads zero volts dc.

3. Release S1. Now, watch the voltage climb back upwards somewhat on your VTVM.

This is an excellent demonstration of 'voltage memory' as it takes a finite time for those little molecular dipoles in the dielectric to realign themselves to the new electric field.

WRITING A D/A CONVERSION PROGRAM IN BASIC:

Is certainly a simple matter if we solve the RC time constant equation on page 7-2 for T = time in seconds and use a charging voltage of 100 volts dc to the circuit in figure 7-3. The equation then is written as:

T (seconds) = LOG(100/(100-X))

Where 'X' is our D/A converter's desired analog dc voltage. All our BASIC prgram need do is convert the voltage we input in our program to the time in seconds that we wish to charge 'C' through resistor 'R.' IF we have a 2 line loop in our program that accomplishes 80 loops per second, then all the program need do is calculate the number of loops equivalent to the time period = T times 80 and hold the VAR/80 DBO zero relay closed for that amount of charging time, before opening.

Here is the BASIC program for our homebrew D/A converter:

```
10 'D/A CONVERSION PROGRAM FOR TRS-80      -     WRITE1
20 '
30 CLS:DEFINTA-Z:INPUT"DESIRED  ANALOG  VOLTAGE  OUTPUT";X
40 Y=0:T=80*(LOG(100/(100-X)))
50 Y=Y+1:OUT0,1:IFY=TTHENOUT0,0:GOTO70
60 GOTO50
70 PRINT:PRINT"NOW READ ANALOG VOLTAGE ON VTVM"
80 PRINT:INPUT"PRESS ENTER FOR ANOTHER SEQUENCE";INPUTR
90 FORZ=1TO24000:OUT0,2:CLS:PRINT"ZEROING OUT 'C'":NEXT
100 OUT0,0:GOTO30
```

This program is self-explanatory except for lines 90 & 100.

Line 90: Introduces a one minute delay to allow the VAR/80's DBO 1 relay to 'bleed' down the voltage and then depolarize 'C' before making another conversion. The MORE time used, the MORE accurate it will be.

Line 100: Turns 'off' the DBO 1 relay, and initiates another conversion sequence.

This little program and homebrew conversion system leaves a great deal to be desired, but at least it is a beginning and may give you the germ of an idea to create your own design. What it really needs to make it useful is a relatively fast A/D converter in the overall system loop so that ONCE capacitor 'C' is charged to the correct analog value, the A/D converter continually checks its value and PUMPS UP the voltage to hopefully maintain the correct value within a few LSBs.

BASIC is really too SLOW for this crude variety of D/A converter. An assembly language program many hundreds of times faster would be much more appropriate. Also, using electromechanical relays introduces all kinds of timing errors. A high voltage transistor switch would improve accuracy considerably.

The most significant disadvantage of using the RC time constant equation is the exponential curve which will make the lower voltage readings so 'coarse' as to be of little use, UNLESS the device being controlled also responds exponentially. An alternative choice would be to use a 16 bit digital word to control this variety of D/A converter which would improve scalar definition by a factor of 256 and yield acceptable accuracy over most of a 0 to 100 volt dc range. With the TRS-80's 8 bit word per port, this is not easy to accomplish without adding considerable external circuitry to decode and implement TWO separate 8 bit PORT values.

A BIT OF D/A CONVERTER HISTORICAL PERSPECTIVE:

Much like A/D conversion techniques, there are probably as many D/A techniques as there have been D/A converter designers. Most of the major historical highlights in this rapidly developing technology have occurred in the last decade or so and would include:

1967: First 8 bit D/A converter utilizing multiple monolithic integrated circuits and thick film resistor networks, developed by Beckman Instruments.

1968: Ten bit D/A monolithic D/A converter based on 722 opamp requiring external components introduced by Fairchild.

1970: First 8 bit D/A converter manufactured using thin film resistor networks by Micro Networks, Inc.

MULTIPLYING D/A CONVERTERS:

Are fundamentally R-2R ladder D/A converters that may utilize variable reference voltages. The output is the product of the reference voltage AND the value of the input digital word. This product is said to be in a single quadrant if the reference voltage is unipolar, two-quadrant if bipolar, and four-quadrant if the two current summing lines illustrated in Figure 7-4, OUT1 and OUT2, feed an operational amplifier that subtracts the difference value between the lines. The D/A network in Figure 7-4 may be used in EITHER the multiplying or non-multiplying mode.

DEGLITCHED D/A CONVERTER:

Is especially important in video display applications where the voltage spikes introduced in the conversion process really 'tear' (literally and figuratively) the video signal apart due to the wide bandwidth and resulting susceptibility to high frequency noise. A simple 'sample and hold' stage fed by the D/A converter's output serves as a smoothing filter (virtually perect) for this noise by holding the last analog voltage value during the conversion process and NOT updating till the conversion is complete and all noise spikes long since gone.

SPECIALIZED HIGH POWER D/A CONVERTERS:

Actually, this heading is a misnomer as ALL D/A converters are of the micro or milli power level variety. What one does with their analog dc voltage output is another matter. If one chooses to use this output to control the output of a 60 amp 220-240 volt ac Triac that drives a steam roller, it could be called a specialized high power D/A converter if you do not wish to use the English language too precisely.

Intersil, now part of Datel, or vice versa, now has an interesting series of dc power amplifiers designed specifically for use with D/A converters for driving dc motors, linear actuators, golf carts, or what-have-you. This series is rated up to 35 volts dc and with proper heat sinking will deliver: IH8510 = 1 amp, IH8520 = 2 amps, and IH8530 = 3 amps on a continuous basic. They are driven by the Intersil 301A op amp ($.50 each) which will mate with most any variety of D/A converter and vary in price from approximately $8. to $12. each. They should see wide acceptance in most every application imaginable of computer control in the 10 to 100 watt power level.

The previous section ONLY covered a few of the myriad D/A conversion techniques that have been used during the last decade. On a percentage basis though, hopefully we have covered the majority of those types that have actually gone into quantity production AND that would be of interest.

- BUILDING SOME WORTHWHILE D/A CONVERSION SYSTEMS -

TRS-80 INTERFACES FOR D/A CONVERSION SYSTEMS:

As mentioned earlier, the Telesis Lab's VAR/80 and Alpha Product's Interfacer 2 make ideal interfaces for tying the TRS-80 into a D/A conversion system whether your particular application is maintaining a constant temperature in your aquarium (the TRS-80 as an expensive thermostat) or guiding an automated vertical boring machine that is refurbishing the valve seats on over-the-hill V-8 cylinder heads in a semi-automated engine rebuilding factory.

Gridley asked, "what about those readers out there in the vast wastelands of the world who do not have a VAR/80 or an Interfacer 2, nor have the price of one? You gonna leave 'em stranded or running for the cookie jar?

No way, Gridley. You have touched a tender spot in our otherwise cold and calculating heart. We will try to run through a do-it-yourself interface to the cruel outside real world that they can build for a few dollars if they wish, plus for an encore, a PORT selector with 8 mini-switches that will allow them to use any TRS-80 PORT from zero to 255 they wish. Let's start by putting together a simple 3 chip interface that will allows us to OUTput data via PORT zero.

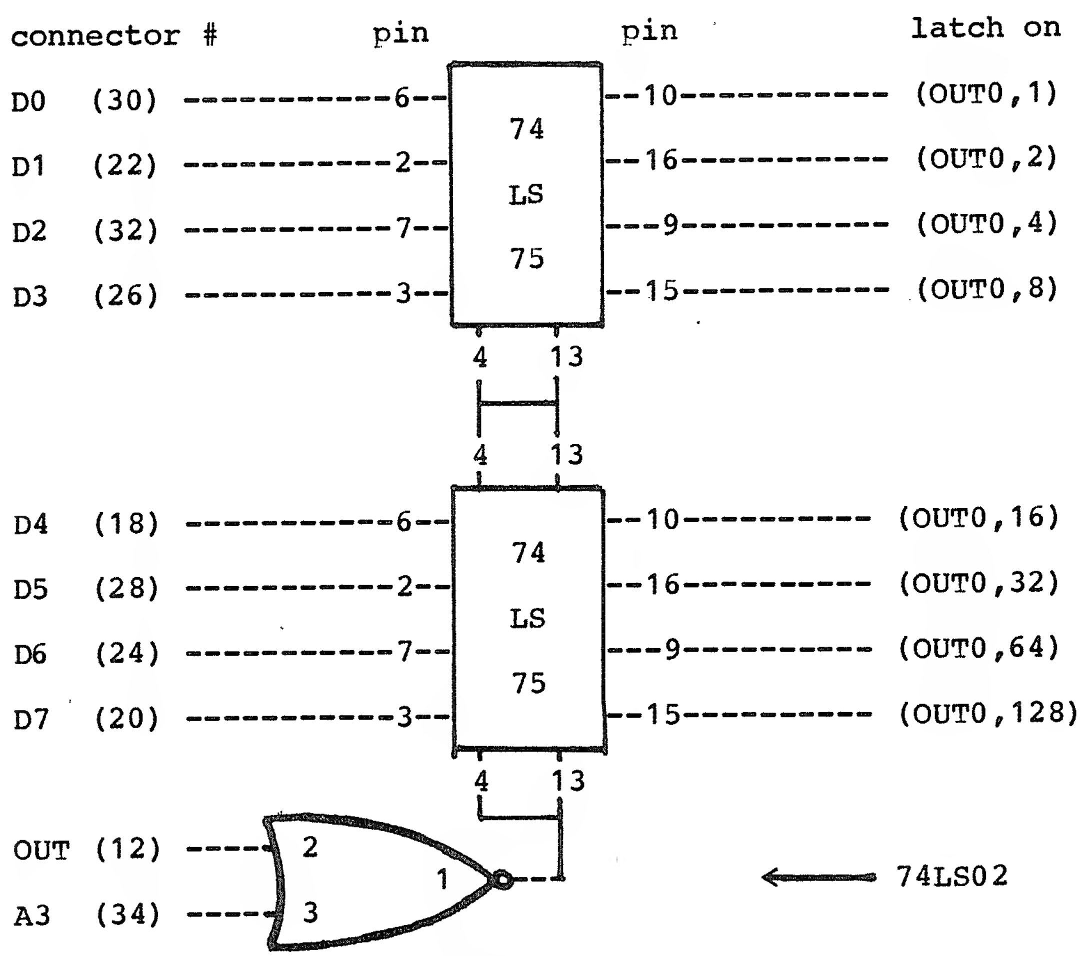


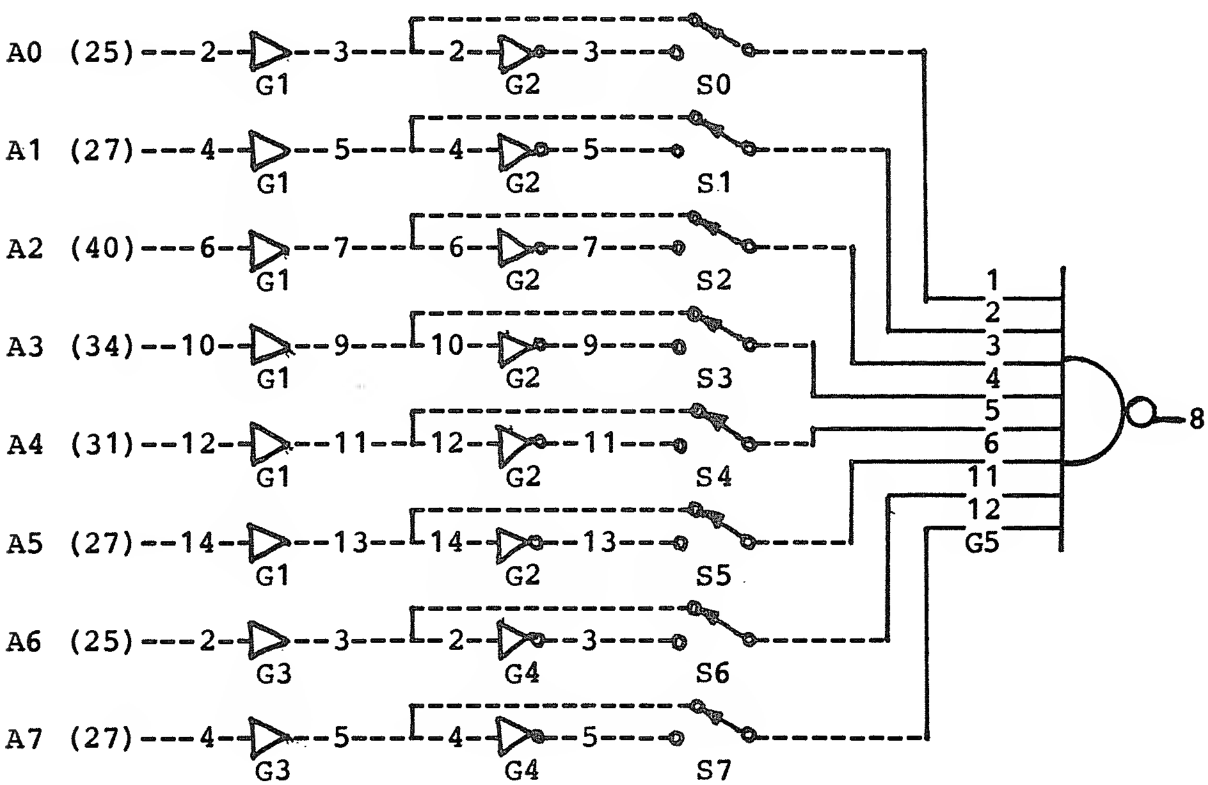
Figure 7-5

Figure 7-5 does NOT show the pinouts for either +5 VDC or ground on the 3 chips. Be sure to ground pin 12 on the 74LS75s and pin 7 on the 74LS04. +5 VDC goes to pin 5 on the 74LS75s and pin 14 on the 74LS04.

The left hand column in Figure 7-5 shows the pin connections to either the keyboard or screen printer port on the expansion interface. The far right hand column shows the eight latches and the instruction that will latch each one individually; i.e., OUT0,1 = 00000001 binary turns 'on' number 1; OUT0,2 = 00000010 turns 'on' number 2; OUT0,4 = 00000100 turns 'on' number 3 and so forth. OUT0,0 turns them ALL 'off' and OUT0,255 turns then ALL 'on.' After an OUT0,xxx statement they will remain 'latched' in that state till the next OUT0,xxx statement.

WHAT IF WE DO NOT WANT TO USE PORT ZERO ? ? ?

You are back, Gridley. Good question. Try this PORT SELECTOR.



G1 & G3: 74LS367 Tri-State hex buffers RS #276-1835
G2 & G4: 74LS368 Tri-State hex inverters RS #276-1836
G5 : 74LS30 8-input NAND gate RS #276-1914
S0 - S7: single-pole single-throw mini-switch RS #275-612

NOTE: RS = Radio Shack part number

Figure 7-6

ELECTRIC PENCIL SURE DRAWS BETTER SCHEMATICS THAN YOU CAN ! !

Michael Shrayer thanks you, Gridley. Let's press on.

To hookup our new PORT selector to the 74LS02 and dual quad latches in Figure 7-6, thus allowing us to select ANY port we wish, use the circuit shown below in Figure 7-7.

```
OUT (12)--10--|>--9-----+-------------------> pin 2 74LS02
              G3        |
                        L------> pins 1 & 15 G1 & pin 1 G3

Tie pin 15 of G3 AND pins 1 & 15 of G2 & G4 ALL to ground

Pin 8 of G5--------10--|>o--9---------------> pin 3 74LS02
                       G4
```

Figure 7-7

What we have done here is to isolate (as in high impedance) the address bus outputs A0 to A7 UNTIL the OUT line goes low, thus avoiding any loading problems by our PORT selector. The address bus does not even know we are there till an OUT statement comes along and turns 'on' the eight buffers.

Switches S zero through S seven in Figure 7-6 are shown in the positions where OUT255,xxx would be decoded. IF we move S7 to the lower switch postion, then the bottom line inverter, G4 would transpose the signal OUT254,xxx = 11111110 binary on the address bus to a 11111111 binary which would turn 'on' G5.

SONOFAGUN, WE DECODED ANOTHER PORT. WHAT ABOUT THE REST ? ?

Hang in there, Gridley. IF we moved ALL the switches, S zero through S seven to their lower postion, then an OUT0,xxx would be inverted to a 'low' to G5 which would then go 'high' thus decoding PORT zero. As such, by setting switches S zero to S seven we may decode ANY PORT from zero to 255. Forinstance:

PORT SELECTION	SWITCH POSITIONS U=UP D=DOWN							
	7	6	5	4	3	2	1	0
0	D	D	D	D	D	D	D	D
1	D	D	D	D	D	D	D	U
2	D	D	D	D	D	D	U	D
3	D	D	D	D	D	D	U	U
4	D	D	D	D	D	U	D	D
5	D	D	D	D	D	U	D	U
6	D	D	D	D	D	U	U	D
7	D	D	D	D	D	U	U	U
8	D	D	D	D	U	D	D	D

SOMETHING SURE LOOKS FAMILIAR ABOUT THE PORT SELECTION CHART. WHAT 'IF' WE CHANGED THE D's to zeroes and U's to ones ? ? ?

Brilliant idea, Gridley. You've GOT it for ALL 255 in binary.

- NOW, LET'S BUILD SOME D/A CONVERSION SYSTEMS -

Hopefully, we have solved the OUT PORT interface problem. The following circuits will work with the VAR/80, Interfacer 2, Figure 7-5's 3 chip design, and for those desiring the BEST of all possible worlds, Figure 7-6's universal PORT selector that may be used with ANY of the foregoing three systems.

IF you are using the VAR/80 or Interfacer 2, you must tie a +5 VDC source to DBO 0's and DBO 1's relays NC = normally closed contacts through a 2000 or 3000 ohm 1/4 watt resistor to drive our D/A converters. ALSO, you MUST tie DBO 0's and DBO 1's NO = normally open relay contacts to ground.

Our homebrew OUT interface converter in Figure 7-5 already provides a nominal +5 VDC 'high' on ALL the latched outputs when activated. We will use the 'C' center contacts on the VAR/80 and Interfacer 2 for DBOs 0 and 1, plus the DBO terminal strip outputs DBO 2 through 7 for the rest.

IF you wish, ADD a far right column on the margin of the page to Figure 7-5, labeling "DBO 0" to the right of OUT0,1; "DBO 1" to the right of OUT0,2; and continue down the column with "DBO 7" to the right of OUT0,128 as from now on we will refer to all output terminals in the format of DBO 0, 1, 2, 3, etc.

THE NATIONAL SEMICONDUCTOR DAC0808 8-BIT D/A CONVERTER:

Is illustrated in Figure 7-8, courtesy of Natl. Semiconductor.

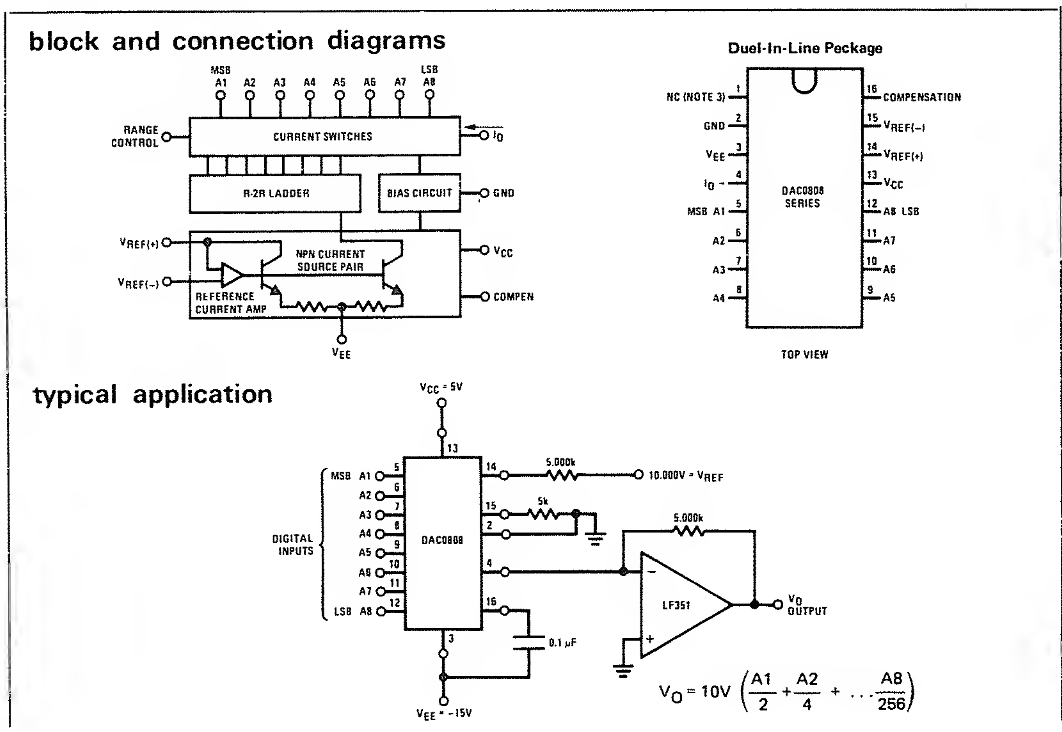


Figure 7-8

This extremely cost effective D/A converter and the sundry parts we will be using are available from: Digi-Key Corp., Box 677, Thief River Falls, Minnesota 56701 or phone toll free: 800-346-5144 and order: (our parts arrived in 3 days)

Part Number	- Description	- Price
DAC0808LCN	8-bit D/A converter .19% linearity	$ 3.37
LF351N	Bi-FET general purpose op amp	.78
LM341P-5	+ 5 volt 0.5 amp pos. volt. regulator	1.04
LM341P-10	+10 volt 0.5 amp pos. volt. regulator	1.04
LM341P-14	+15 volt 0.5 amp pos. volt. regulator	1.04
LM320MP-15	-15 volt 0.5 amp neg. volt. regulator	1.74
	total investment =	$ 8.98

The DAC0808 is of the R-2R variety and may be used in or out of the multiplying mode. If used in the multiplying mode its output may vary over the range of 1 to 256:1 by varying the input current to reference terminal 14 from 16 microamps to 4 milliamps (= 256 X 16 uA).

Figure 7-9 illustrates the equivalent circuit of the DAC0808.

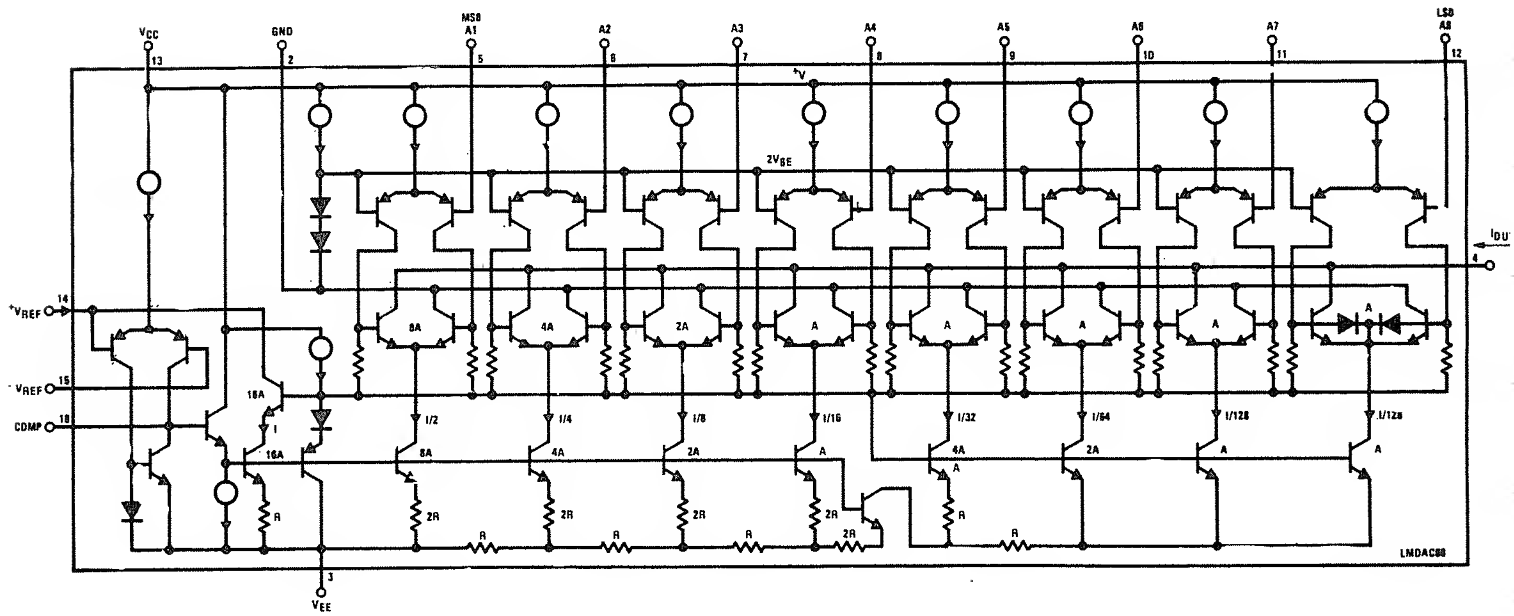


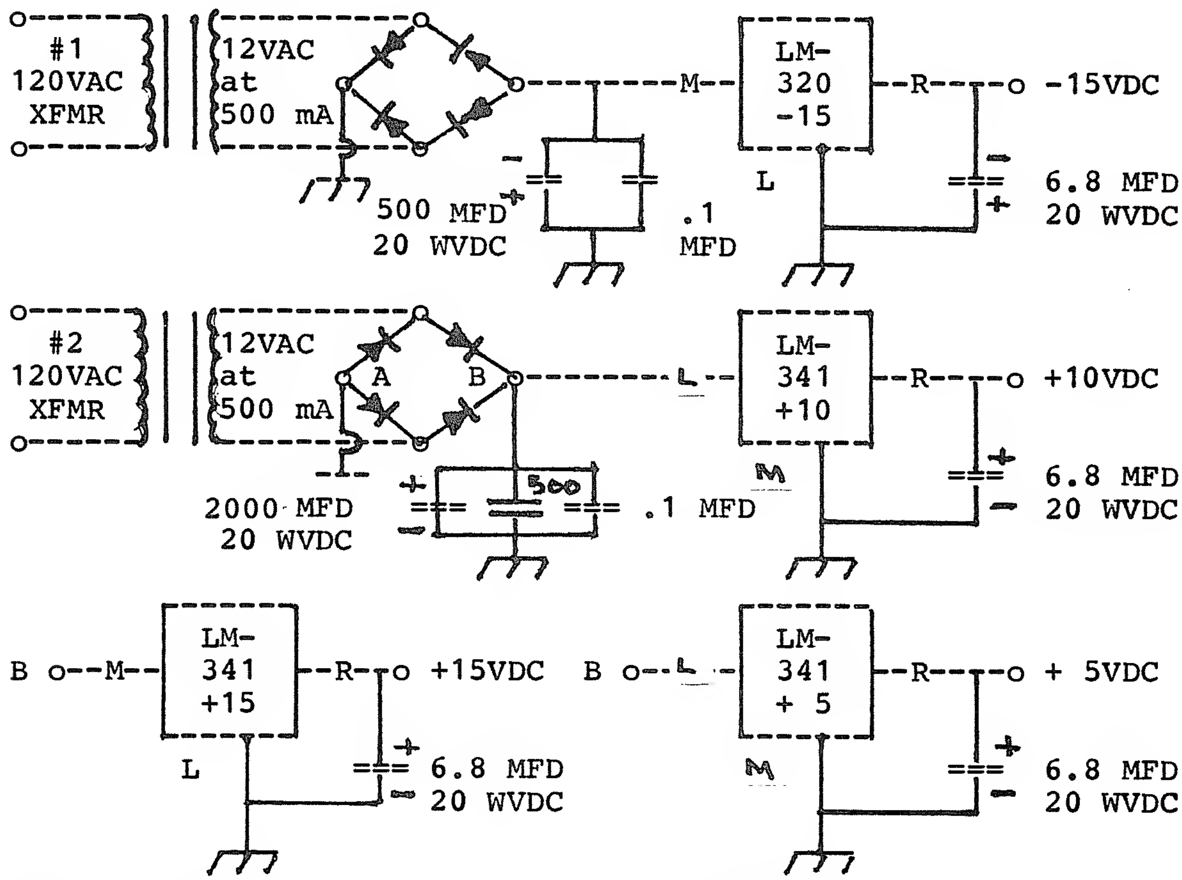
Figure 7-9

The R-2R ladder network is somewhat similar to that shown in Figure 7-4. When used in the multiplying mode (as we will use it), the slew rate (speed) = 8 milliamps/microsecond, and that is truly FAST. Full scale current match = + or - 1 LSB and settling time = 150 nanoseconds.

Let's build our converter on a piece of perfboard with modestly short, say 12" to 15" leads connecting it to the

VAR/80 or the interface of your choice. IF you build your own interface, you cannot beat the Hobby World TRS-80 40 pin edge connector and 36" long 40 conductor cable @ $4.75 each. They are part number 1980 and available from Hobby World, 19511 Business Center Drive, Northridge, California 91324.

For a power supply, we used two 12 volt @ 500 mA 110/120 VAC transformers hooked up as shown in Figure 7-10. The peak voltage output of these transformers were 1.4 X 12 = 16.8 which is adequate to obtain + and - 15 VDC from these regulators.



NOTE: voltage regulators L=left, M=middle, R=right terminals.
6.8 MFD caps are tantalum; others electrolytic/ceramic.

Figure 7-10

There are all sorts of ways to build this power supply as the current requirements are practically nil, except for the + 5 VDC segment which only has to furnish 100 to 200 milliamps. Figure 7-10 is only ONE way to go about it. The silicon rectifiers many be most any variety of 20 volts PIV or better. The second 500 MFD electrolytic shown is optional, plus you probably could get away with using a single transformer IF its current capacity was adequate.

Either Poly Paks or Hobby World are good sources for any parts not available in your electronics 'junk' box. We have used Poly Paks for years and MOST of their surplus parts are good.

This little power supply will handle BOTH the homebrew interface systems in Figures 7-5 and 7-6 as well as Figure 7-8' D/A conversion system. It also serves to power the LF351 op amp.

FOUR important items to note BEFORE soldering up the D/A converter and LF351 op amp.

1. The two 5.000 K ohm resistors shown in Figure 7-8 should be 1 percent OR BETTER tolerance resistors. IF you have an accurate digital volt-ohmmeter, by all means make them yourself; i.e., filing a notch in a nominal 4.7K ohm resistor till you reach 5K ohms. The 5K ohm resistor going to pin 14 'sets' the multiplying range of the converter, so MUST be accurate. The 5K ohm resistor beween the LF351 op amp output and minus input 'sets' the amplification factor of the FET op amp, so MUST be accurate too.

2. Connections to the LF351 FET op amp are as follows:
   A. Pin 2 inverting input (- minus) to pin 4 of the DAC.
   B. Pin 3 non-inverting input (+ plus) to ground.
   C. Pin 6 output and feedback resistor connection.
   D. Pin 7 V+ to + 15 VDC regulated power supply.
   E. Pin 4 V- to - 15 VDC regulated power supply.

3. The LF351 will supply full voltage swing to a load of 2000 ohms or MORE. To be on the safe side, add a 4.7K ohm resistor between pin 6 and ground and do NOT expect a barefoot LF351 to directly drive any low impedance devices WITHOUT a suitable amplifier such as the Intersil IH8510-20-30 mentioned earlier.

4. ADD a 100 pF ceramic disk capacitor in parallel with the 5K ohm resistor between the LF351's pins 2 and 6 to eliminate any potential parasitic problems.

-CONSTRUCTION SUGGESTIONS:

-Use a #60 bit to drill the holes in the perfboard for the DAC0808's 16 pin DIP socket and LF351 8 pin DIP socket.

-KEEP leads as short as possible and avoid capacitive coupling in the layout IF you wish to use it in high speed applications of any variety. At the same time, do not try to overcrowd your perfboard so it is difficult to work on. In other words, just follow GOOD standard construction practices. A good looking board usually works the same way.

-Install the 5.000K resistor that goes to pin 14 of the ADC0808 so it may be easily removed and replaced a little later with a 4.5K ohm resistor when we modify this board for the ADC0800.

-Most ANY 12 VAC transformer will put out considerable MORE than the nominal 12 volts times 1.4 = 16.8 peak voltage at these very LOW current loads. Of the 3 tested, mostly of 1 amp variety, ALL managed to put out 19 to 20 volts dc.

TROUBLE SHOOTING TECHNIQUES:

Prototype board trouble shooting has it own special guidelines that should be followed whether it is only a little two or three chip job on perfboard or a 100 chip multi-layer monster. IF your joy and delight does NOT work when first turned "on," try these gambits.

1. After 35 years of homebrew construction, the author's most common error is leaving out a coupling line/connection or component. The simple and obvious solution to this 'stupid' trick is to use a red pen or pencil and trace out the connection/component on the schematic AFTER it is soldered in place.

2. The next most common error is soldering a connection to the WRONG DIP terminal pin on the back of the perfboard. After your DIP sockets are installed on the perfboard, use a fine felt tip marking pen to mark the four corner terminal numbers adjacent to pins 1, 8, 9, and 16 ON THE BACK OF THE BOARD. Be glad that we are not using 40 or 48 pin DIP sockets. When you THINK you are all done, go back and check the UNUSED DIP pin terminals. Are they the correct ones?

3. Color coded 1/4 watt resistors are easily misread with the resulting wrong value installed. We have watched a totally color blind technician install hundreds/thousands of color coded resistors per day with NEVER an error. She used properly labeled parts bins for each value.

4. Using a 3 inch magnifying glass, check your work for clean and bright solder joints (cold solder joints are usually a dull gray). Check for solder 'hair' shorts that can spoil your whole day. Due to the close proximity of DIP pins, they are usually found there.

5. MOST IMPORTANT of all. Use the right 'heat' soldering pencil for the job at hand, NOT a soldering gun. Most amateurs do NOT realize it, but THREE heating elements are usually required for a SINGLE job IF you wish to do professional quality work: a 27 watt Ungar heating element with #PL-338 silver plated IRON CLAD tip, a 42 watt Ungar heating element with the same tip, and a 50 watt Ungar combined heating element/tip, #4036S that is pyramid shaped and silver plated. Also, always use Ersin Multicore or Kester .031" 60/40 solder. More amateur construction projects are SPOILED by using the wrong soldering iron and/or wrong solder than another other single cause.

OUTPUT LEVELS OF THE DAC0808 CONVERTER AND LF351 OP AMP:

As configured in the schematic in Figure 7-8, our system should put out a nominal analog voltage between - 10 VDC and + 10 VDC over the DIGITAL input range of 0 through 255 IF your component values and input voltages are EXACTLY as specified. Since the specs allow + or - 1 LSB error = approx. 80 millivolts, let's test it and see exactly HOW close we have come.

TESTING THE DAC0808/LF351 CONVERSION SYSTEM:

Here's our SECOND most favorite BASIC test program that will allow us to checkout the accuracy/linearity of our new D/A conversion system. It requires an accurately calibrated digital voltmeter that will readout at least to the 1 millivolt level IF you want a true measurement. An ordinary 5 inch scale 20,000 ohms per volt volt-ohmmeter may be used on the 3 VDC and 15 VDC ranges with somewhat less accuracy.

This program assumes that you have a 'latching' output for PORT zero whether it be the homebrew variety in Figure 7-5, a VAR/80, or an Interfacer 2 to drive the converter. It is quite tolerant of the relative accuracy of the value of the 5.000K ohm resistor to the DAC0808 pin 14 that determines the multiplication factor of the R-2R converter, as it allows YOU to input the 'GO' minus voltage output for OUT0,0. Ideally 'GO' should be -10.000 VDC, but such is not always the case.

Unfortunately this system and program is not so tolerant regarding the accuracy of the LF351's 5.000K feedback resistor, so try to 'file' this one 'right on the money.'

```
10 'DIGITAL TO ANALOG CONVERTER TEST PROGRAM     -     DAC2
20 '
30 CLS:INPUT"INPUT INITIAL OUT0,0 VOLTAGE READING ";B:PRINT
40 K=B:B=SQR(B*B)*2
50 PRINT"PORT ZERO DIGITAL/ANALOG TEST - INPUT ANALOG READING"
60 A=B/255:FORD=0TO255:C=K+(A*D):OUT0,D
70 PRINT"OUTPUT SHOULD = ";INT(C*1000+.5)/1000;"VOLTS DC";
80 PRINT"......IT ACTUALLY = ";:INPUTE
90 F=C+.08:G=C-.08: REM USE VARIABLE 'A' IF DESIRED
100 IFF<ETHENPRINT"OUT OF RANGE BY";:GOTO140
110 IFG>ETHENPRINT"OUT OF RANGE BY";:GOTO150
120 PRINT"OK"
130 PRINT:NEXT
140 PRINTINT((E-F)*1000+.5)/1000;"VOLTS DC HIGH":GOTO130
150 PRINTINT((G-E)*1000+.5)/1000;"VOLTS DC LOW":GOTO130
```

Let's take this mini-program apart, BRIEFLY.

Line 30: allows the program to start at an OUT0,0 NEGATIVE analog output voltage other than MINUS 10.000 VDC via INPUT.

Line 40: variable 'K' remembers the intitial value of 'B' and the rest just converts 'B' to a positive number without going through the SGN repertoire, plus multiplying 'B' times 2.

Line 50: is a message to remind us what is going on.

Line 60: divides 2 times our starting voltage into 255 increments, starts a 0 to 255 FOR-NEXT loop, multiplies the loop value times the 1/255 increment and adds it to 'GO,' and lastly outputs the loop value via PORT zero for conversion.

Line 70: Reminds us of what the theoretical value of our analog output should be with an accuracy to 1 millivolt.

Line 80: then asks us to INPUT via the keyboard the dc voltage output by our D/A converter as read on our digital voltmeter.

Line 90: sets the plus or minus one LSB limits for our D/A converter's output IF we started at - 10VDC. Change it as desired for other ranges. Obviously, variable 'A' may be substituted for both .08 values IF you wish.

Lines 100 & 110: test the 'ideal' voltage against the plus or minus one LSB error limits and send the program off to lines 140 and 150 IF these limits are exceeded.

Line 120: IF the voltage limits PASS line 100's & line 110's tests, thereby falling through to this line, then in FOURTH language style (note spelling) it prints 'OK.' (That's a joke, Gridley. Have you ever counted your thid, forth, and fith fingers?)

NO THIR. JUST MY ONETH AND TWOTH.

Line 130: skips a line and increments our FOR-NEXT loop which sends the program back to the middle of line 60's FOR statement to intitiate a look at the next incremental dc voltage.

Lines 140 & 150: printout the error voltage GREATER than or LESS than one LSB error to one millivolt accuracy, and then send the program back to line 130 for the next increment.

COMMENT:

Microsoft BASIC is a 'super' language for electronics laboratory programming IF you use it wisely.

Most of the bull roar one hears about FORTRAN being the 'only way' for efficient lab test programming is due SOLELY to engineers and technicians who have never fully learned and do not understand the extremely broad and versatile capabilities of Microsoft's Level II and/or disk BASIC. Do not try to shame them for their stupidity as you will only make a new enemy.

Just show them what your TRS-80 can DO properly programmed and you will make a new friend. IF you need just a bit more speed, like 50% or 100%, try the Mumford Micro clock speed-up adaptors AND do not be afraid to plug-in a Z-80A IF your original Z-80 will not hack the 100% speed increase.

It requires a 'great deal' of patience and fortitude to teach 'old dogs new tricks,' BUT they can and will learn if you will only persevere. One well known microwave engineering quality control test lab NOW has 3 TRS-80s replacing 3 WANG mini-computers that cost 10 times as much and MOST IMPORTANTLY, are more versatile and do a better job than their predecessors.

This Chapter is running a bit longer than originally planned, so let's jump into the next D/A lab experiment that will only require a few modifications to the perfboard you built for the DAC0808. It utilizes the DAC0800 which is considerably faster than the DAC0808, plus offering differential output voltages (up to + and - 20 VDC) WITHOUT an op amp. Price of this excellent National Semiconductor D/A converter is only $3.87 from the Digi-Key Corporation.

THE NATIONAL SEMICONDUCTOR DAC0800 8-BIT D/A CONVERTER:

Is quite similar to the DAC0808 as shown in the block diagram and pinout connection diagram in Figure 7-11.

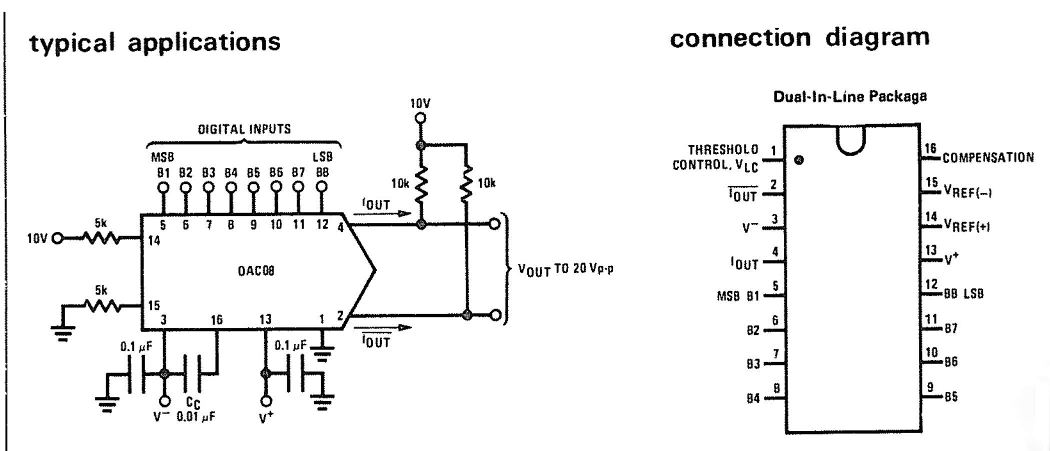


FIGURE 1. ±20 Vp-p Output Digital-to-Analog Converter

Courtesy Natl. Semiconductor

Figure 7-11

We will start out with the basic circuit in Figure 7-11 plus the following modifications to allow us to accurately 'set' the full scale output voltage level.

1. Change the 5K ohm resistor from pin 14 to the +10 VDC reference to a 4.5K resistor. Check a few 20% 4.7K ohm resistors in your junk box and you will probably find one VERY close to the 4.5K ohm value.

2. Add one END terminal of a 50K ohm potentiometer to the junction of the +10 VDC reference AND the 4.5K ohm resistor that goes to pin 14. Solder the other end of this pot to ground and connect the center terminal to pin 15. DO NOT install the 5K ohm resistor to pin 15 shown in Figure 7-11.

In addition to #1 and #2, note the following pin connection and component changes from the DAC0808 hookup:

- Pin 1 to ground.

- Pins 2 and 4 EACH to a 10K ohm resistor to +10 VDC and pins 2 and 4 EACH to a solder lug to allow us to attach either a digital voltmeter of 20,000 ohms per volt volt-ohmmeter.

- Pin 3 still goes to our -15 VDC regulated reference source, BUT change the .1 MFD disc that went from pin 3 to pin 16 TO from pin 3 to ground. ADD a .01 MFD disc ceramic from pin 3 to pin 16.

- Pins 5 to 12 are still attached to our latching interface's output. Note that pin 5 is the MSB and pin 12 the LSB. Therefore, pin 5 goes to DBO terminal 7, pin 6 to DBO 6, pin 7 to DBO 5, and so forth with pin 12 to DBO terminal 0.

- Pin 13 now goes to the +15 VDC voltage reference, plus ADD a .1 MFD disc ceramic from pin 13 to ground.

- This chip does NOT require the +5 VDC regulated power supply used with the DAC0808. Only the + & - 15 VDC and +10 VDC regulated supplies are necessary.

- Use a 10 or 20 turn 'trim pot' for the 50K ohm 'full scale' adjustment. It will make full scale adjustment MUCH easier.

- The accuracy of the + and - 15 volt regulators' output is important as well as the EXACT value of each 10K resistor IF you wish PERFECT centering; i.e., OUT0,127/128 = 0 VDC.

LET'S HOOK IT UP AND PLUG IT IN:

Be sure to check the polarities of each power supply lead BEFORE plugging them into the DAC0800 to avoid $3.87 mistakes. Now, lets start with an OUT0,255 IN BASIC.

What happened, Gridley?

S-M-O-K-E IS WHAT HAPPENED AND I CHECKED POLARITIES TOO ! ! !

Sorry about that, Gridley. You have your DAC0800 plugged into the 16 pin DIP socket backwards. It will fit into the socket both ways, so best check to see you have it properly oriented next time. Here's another DAC0800. Let's start all over again.

What happened this time, Gridley?

MY VOLTMETER PINNED THE NEEDLE BACKWARDS ! ! !

Ok, Gridley. Kindly reverse the voltmeter terminals and let's try it one more time. Remember to OUT0,255 in BASIC.

I am afraid to ask, but what happened on this try, Gridley?

SONOFAGUN, I THINK ITS WORKING....IT READS +19 VDC. WHAT DOES THAT MEAN ? ? ?

It means that we have to adjust the 50K ohm 'trim pot' for the FULL SCALE reading we desire. Let's try to set it up for .100 amp per bit. Adjust the trim pot for about +13 VDC output.

In the most 'perfect' of all possible worlds, our + and - 15 VDC voltage regulators would actually output EXACTLY 15.000 volts dc, but since this is NOT a 'perfect' world, our voltage regulators' tolerances are + and - 600 millivolts which is a WHOLE BUNCH, so we should expect SOME minor errors.

What were are shooting for is a 'step' of .100 volts per bit output to PORT zero. With our particular setup, this was achieved by adjusting the trim pot to exactly +13.2 VDC with an OUT0,255 which gave us (luckily) -12.4 VDC for an OUT0,0. Since our 10K ohm resistors were only of the 5% tolerance variety (gold band), and the + 15 VDC regulator put out 15.4 VDC, we were fortunate indeed that the 'zero VDC crossing' was at OUT0,123 which was ONLY 3 bits away from a perfect OUT0,126 or 127.

Here is what our ADC0800 system output CORRECTED for + or - one LSB = .1 volts dc which is the tolerance of the ADC0800:

OUT0, VALUE	VOLTAGE DC	OUT0, VALUE	VOLTAGE DC
0	-12.4	9	-11.4
19	-10.4	29	- 9.4
39	- 8.4	49	- 7.4
59	- 6.4	69	- 5.4
79	- 4.4	89	- 3.4
99	- 2.4	109	- 1.4
119	- .4	120	- .3
121	- .2	122	- .1
123	.0	124	+ .1
125	+ .2	126	+ .3
127	+ .4	129	+ .6
139	+ 1.6	149	+ 2.6
159	+ 3.6	169	+ 4.6
179	+ 5.6	189	+ 6.6
199	+ 7.6	209	+ 8.6
219	+ 9.6	229	+10.6
239	+11.6	249	+12.6
254	+13.1	255	+13.2

This a remarkable performance considering the 'sloppy' tolerance components we used in this test. IF you allow one LSB error (which we did) for a poorly calibrated (after dropping it) voltmeter, plus one LSB error allowed for the ADC0800, it is still an outstandingly accurate digital to analog converter for the $ 3.87 price. By itself, ADC0800 applications are only limted by your imagination where modestly high impedances are involved. The Natl. Semiconductor ADC0800 data sheets cover most every configuration imaginable, including operational amplifier interfaces for ALL these configurations.

WISH TO DIG DEEPER INTO D/A AND A/D CONVERTER TECHNOLOGY:

Probably the BEST and certainly the most COST EFFECTIVE book on the subject is published by DATEL, 11 Cabot Blvd., Mansfield, MASS. 02048 and entitled, "Data Acquisition and Conversion Handbook. It is availbale at $3.95 postpaid (U.S.).

- CHAPTER 8 -

A DIFFERENT APPROACH TO WRITING A MORSE CODE PROGRAM

We are grateful to Col. Charles D. House, USAF (Retd) the winner of the "Fastest Footrace Contest" for allowing us to reprint the FIRST vestiges of an updated and potentially faster Morse Code program than the one presented in Volume 2's Chapter 10.

It utilizes an approach that Chick used to 'win' the Fastest Footrace Contest by nearly 50% over the next competitor. This program has the potential of transmitting up to and exceeding 100 words per minute in BASIC when lines 76 & 77 are changed.

Though Chick's program is FAR from finished as we go to press with Volume 3, we included it BECAUSE there are a number of excellent approaches and ideas that YOU may use to speed up BASIC programs of MOST any variety.

Study the two programs, and note the considerable differences between Volume 2's Chapter 10 and this one by Chick House IN THE TRANSMIT SECTION. There are many ways to skin a cat (forgive me R/C and Harlequin), but Chick's is certainly the fastest we have yet seen, and most probably the FASTEST that can be accomplished without using BASIC Compilers and all that foolishness, short of an efficient assembly language program which would OF COURSE beat all other competitors hands down.

IF the 'rules of the game' are pure BASIC (and they were), then Col. House's approach WINS again.

OUR collective 'hats off' (doff your hat too, Dave Lien), to Chick House for showing us 'would be' speedsters the way. Though the 'Disassembled Handbook' series is about assembly language programming, using ALL THOSE wonderful ROM subroutines available to us that we bought from Paul Allen and Bill Gates of Microsoft, (via purchasing Level II ROM), we CAN ALWAYS pause and learn from a real MASTER. Thank you, Col. House.

REMEMBER:

This program was ONLY partially completed when Volume 3 went to press, and is presented for reference ONLY. It is NOT meant to be or represented as a COMPLETE working program.

```
49 CLS:PRINTCHR$(23):PRINT:PRINT"   TRS-80 MORSE CODE SYSTEM":PR
INT:PRINT:PRINT
50 PRINT" NO ANCILLARY DEVICES REQUIRED":PRINT:PRINT:PRINT"
BY  COL. C. D. HOUSE":PRINT:INPUTR:CLS
51 CLEAR2550:DEFINTA-Z:DIMA$(100),A0$(100),X$(25),D$(25),AF$(100
)
52 DATA12121,,11111111,111212,,,,,,221122,21112,,121212,21121,22
222,12222,11222,11122,11112,11111,21111,22111,22211,22221,222111
,212121,,,,112211
53 DATA12,2111,2121,211,1,1121,221,1111,11,1222,212,1211,22,21,2
22,1221,2212,121,111,2,112,122,1112,2112,2122,2211,8,9,10,31,91
54 FORI=35TO90:READA0$(I):NEXT:FORI=1TO5:READA0(I):NEXT:FORI=1TO
7:READS(I),S1(I):NEXT:FORI=1TO25:READX$(I):NEXT:FORI=1TO25:READD
$(I):NEXT:FORI=1TO100:READA$(I):NEXT
55 DATA5,140,10,80,15,40,20,30,25,20,30,15,35,10
56 CLS:INPUT"ALPHANUMERICS OR MORSE ON VIDEO DISPLAY (A/M)";AA$:
CLS:INPUT"CODE SPEED 5, 10, 15, 20, 25, 30, OR 35 W.P.M.   ";S:C
LS
57 FORI=1TO7:IFS=S(I)THENSI=S1(I):I=7
58 NEXT:S=SI:PRINT@195,"REMEMBER ";CHR$(125);"-- FOR INSTRUCTION
 SUMMARY IN TRANSMIT MODE":INPUTR:CLS
61 PRINT@195,"REMEMBER THE 'CLEAR' KEY IS YOUR T/R SWITCH";:INPU
TR:CLS
64 ONERRORGOTO65
65 RESUME66
66 PRINT"TRANSMIT MODE    ";CHR$(91);" = SIGNAL/MESSAGE CMDS"
67 Z=100:M$=""
68 A$=INKEY$:IFA$=""THEN68
69 IFA$=" "ANDLEN(D$)<1THENPRINT" ";:GOTO67
71 FORI=1TO5:IFA0(I)=ASC(A$)THENII=I
72 NEXT:ONIIGOTO171,170,169,168,79
73 M$=A0$(ASC(A$)):GOTO75
74 IFM$=""THEN67
75 IFAA$="A"THENZ$=A$ELSEZ$=M$+" "
76 FORK=1TOLEN(M$):C$=MID$(M$,K,1):IFC$="2"THENC=S*3ELSEC=S
77 OUT255,4:FORA=1TOC:NEXT:OUT255,0:FORA=1TOS:NEXT:NEXT:PRINTZ$;
:FORA=1TOC:NEXT
78 IFZ=0THENNEXTELSEIFZ=100THEN67ELSEIFZ=1313THEN165ELSEIFZ=1306
THEN163ELSEIFZ=1310THEN164
79 II=0
83 DATACQ,QTH,QRZ,QSL,RST,73,QRM,QRN,QRS,QRQ,QRX,QRT,QSB,QSY,QSY
+,QSY-,CQSS,QRZSS,SECT,TEST,CODE1,CODE2,CODE3,SPEED,CHICK
85 DATA"CQ CQ CQ DE W4UCH/2 K ","QTH IS BOX 1065, CHAUTAQUA LAKE
, N.Y.  14722","QRZ QRZ QRZ DE W4UCH/2 K ","QSL QSL GD LUK IN SS
. DE W4UCH/2 K "
87 DATA"RST 599 599 IN WNY WNY DE W4UCH/2 K ","73 TO U AND URS.
HP TO WK U AGN SOON. DE W4UCH/2 K ","QRM QRM PSE TRY AGN. DE W4U
CH/2 K ","LOCAL QRN QRN PSE TRY AGN. DE W4UCH/2 K "
89 DATA"QRS. PSE SLOW DOWN A BIT. DE W4UCH/2 K ","QRQ. MAY I SPE
ED UP? DE W4UCH/2 K ","QRX. THE PHONE PSE STANDBY. ","MUST QRT N
OW. 73, DE W4UCH/2 K "
91 DATA"QSB TERRIBLE PSE TRY AGN. DE W4UCH/2 K ","QRM WHERE YOU
WISH TO MOVE? DE W4UCH/2 K ","QSY UP 3KHZ UP 3KHZ NOW. DE W4UCH/
2 K ","QSY DOWN 3KHZ 3KHZ NOW. DE W4UCH/2 K "
```

```
93 DATA"CQSS CQSS CQSS DE W4UCH/2 K ","QRZ SS QRZ SS QRZ SS DE W
4UCH/2 K ","WNY WNY WNY DE W4UCH/2 K ","PARIS PARIS PARIS PARIS
PARIS PARIS PARIS PARIS PARIS PARIS "
94 DATA,,,,"WE ARE GRATEFUL TO COL. HOUSE FOR THIS ILLUSTRATION
"
100 DATAE,T,I,A,N,M,S,U,R,W,D,K,G,O,H,V,F,-,L,-,P,J,B,X,C,Y,Z,Q,
-,-,5,4,-,3,-,-,-,2,-,-,-,-,-,-,-,1,6,-,/,-,-,-,-,-,7,-,-,-,8,-,
9,0,-,-,-,-,-,-,-,-,-,-,-,-,?
101 DATA-,-,-,-,-,-,-,-,.,-,-,-,-,-,-,-,-,-,-,-,-,-,-,-,-,-
126 CLS:PRINT"   Q SIGNAL - MESSAGE - SUBCOMMANDS ":PRINT:PRINT"
     JUST ENTER THE NUMBER ....... PLEASE":PRINT:X1=1
127 FORI=X1TOX1+3:PRINTI;"= ";X$(I),:NEXT:IFI=25THEN128ELSEX1=X1
+4:GOTO127
128 PRINTI;"= ";X$(I):PRINT:PRINT"<ENTER> WILL RETURN TO LIST":P
RINT@40,"";:INPUT"SIGNAL/MESSAGE #";M:Z=0:E=-1:CLS:IFM=0THENM=26
129 IFM<21THEND$=D$(M):GOTO159
131 ONM-20GOTO133,134,135,136,137,138
132 GOTO126
133 Z=1313:GOTO165
134 Z=1310:GOTO164
135 Z=1306:GOTO163
136 S=0:INPUT"NEW CODE SPEED";S:GOTO57
137 D$=D$(M):GOTO159
138 D$="AA":GOTO126
159 FORL=1TOLEN(D$):A$=MID$(D$,L,1):IFL=LEN(D$)THEND$="":GOTO67
160 IFA$=" "THENPRINT" ";:FORA=1TO7*S:NEXT:GOTO78
161 GOTO69
162 REM"CODE PRACTICE"
163 D=RND(47)+43:IFD<65ANDD>59THEN163ELSE166
164 D=RND(42)+47:IFD<65ANDD>57THEN164ELSE166
165 D=RND(26)+64
166 A$=CHR$(D):E=E+1:IFE>4THEN167ELSE69
167 FORF=1TO7*S:NEXT:PRINT" ";:E=0:GOTO69
168 REM
169 INPUT"CHARACTER TIMING: 5 NOMINAL";CC
170 INPUT"SPACE TIMING: 3 NOMINAL";SS:GOTO173
171 PRINT"RECEIVE MODE":CC=5:SS=3
173 A=INP(255)
174 C$=INKEY$:IFC$="P"THEN169
175 IFC$=CHR$(31)THEN66
176 IFC$=CHR$(10)THEN209
177 IFA=127THEN173
178 B=0
179 IFA=255THENOUT255,0
180 A=INP(255):B=B+10
181 IFA=127THENC=((5*C)+(2*B))/6:DO=2*DO:DA=2*DA:DO=DO+1:GOTO188
182 IFB<(.5*C)THEN179
183 DO=2*DO:DA=2*DA:DA=DA+1
184 IFA=255THENOUT255,0
185 A=INP(255):B=B+10
186 IFA=255THEN184
187 C=((4*C)+B)/5
188 B=0
189 IFA=255THENOUT255,0
```

```
190 A=INP(255):B=B+CC
191 IFA=255THEN178
192 IFB<(.5*C)THEN189
193 GOSUB199
194 A=INP(255):B=B+SS
195 IFA=255THEN178
196 IFB<(2*C)THEN194
197 PRINT" ";
198 GOTO173
199 DA=DA*2
200 D=DA+DO
201 IFD>100THEND=100
202 PRINTA$(D);
203 DA=0:DO=0
204 RETURN
209 PRINT"AUTOMATIC FILE SUBROUTINE - ENTER WILL RETURN TO TRANS
MIT MODE. EACH FILE MAY CONTAIN UP TO 255 CHARACTERS.":PRINT
210 FORI=1TO100
211 IFAF$(I)<"1"THENINPUTAF$(I):CLS:GOTO66
212 NEXT
310 PRINT"FILE REVIEW - FOUR FILES PER PAGE:":INPUTR:CLS
311 X1=1
312 FORI=X1TOX1+3:PRINTAF$(I):NEXT:INPUTR:CLS:X1=X1+3:IFX1=>100T
HEN313ELSE312
313 PRINT"END OF FILE - ENTER TO RETURN TO TRANSMIT MODE":INPUTR
:CLS:GOTO66
317 REM INSTRUCTION SUMMARY HERE
318 REM THIS PROGRAM IS FOR DEMONSTRATION OF THE 'TRANSMIT'
319 REM SEGMENT 'ONLY'
358 END
```

CHAPTER 9

- COMPUTER BULLETIN BOARD SYSTEMS -

INTRODUCTION:

Higher vertebrates share one thing in common, besides a spinal column. It is the innate need to communicate with one another. Computer buffs are no different than their fellows who revel in amateur radio communications, learned societies whose communications are only understood by their Fellows, or the Friday Morning Music Club where only Bach, Beethoven, and Brahms are the common language.

The computer buff has spawned the computer bulletin board system which through the courtesy of the near universal Bell Telephone System, (the BEST in the world, bar none) allows him to dial up a computer bulletin board system located in the vast majority of metropolitan centers in the U.S. Ma Bell 'loves' those who use her facilities, especially long distance AND especially after normal business hours. She rewards you for using here system during non-business hours by reducing her rates for long distance often as much as 300 to 400 percent, plus.

Picture if you will, nearly 500,000 TRS-80s, Apples, Pets, and Rinky-Dink 6800s out there in the boondocks in the hands of users just dying to talk to each other. Club meetings, if any, are only once a month. Magazines like Kilobaud, 80 Microcomputing, and 80-US are only published once a month. Even books like this one come out only once or twice a year. WHAT to do in between time? WHY just pickup the phone and dial the local or nearby ABBS, CBBS, Forum-80, MicroNet or The Source computer bulletin board/data system and you've got an instant friend out there willing to converse about all sorts of goodies of interest to YOU.

WHO runs them? The first 3 mentioned above are run mostly by individuals or clubs. They are much like amateur radio repeaters on the 2, 1 1/4, and 3/4 meter amateur bands. In virtually all cases they are now open to anyone and everyone, though this may change shortly as the individuals/clubs that installed them find themselves innundated by the ever present 'squirrel" fringe. When that happens, all they do is change the phone number to an unlisted one, and for a time, that's that. The MicroNet and The Source systems listed above are commercial enterprises that require a modest monthly minimum billing (like your phone), and offer services as far ranging as most anything and everything from Real Estate Listings, stock market reports, AP or UP wire services, or they will even play games with you to while away the time.

"IF you think that program swapping is 'bad news' (we don't) and is a violation of U.S. copyright law, you AIN'T SEEN NOTHING YET," we were told by the owner and operator of a major bulletin board system near Washington, D.C.

While we are on the rather delicate subject of copyright we should at least make clear to our readers our viewpoints on the subject. In essence we believe the ANY law that is NOT enforceable is a BAD law. The majority of the U.S. Congress believed that PROHIBITION was a good law some 50 years ago, or they would not have passed it. Happy to say, it was not in the least bit enforceable, so in due course was wisely repealed by those who had passed it.

As we make our living by writing both programs and books, we have a substantial interest in the subject of copyright. Those who bemoan and wail (lot's do) the widespread copying of copyrighted programs are for the most part those who could not 'program their way out of a wet paper bag.' They use the copyright argument as a CRUTCH to support their justifiably deflated egos; i.e., they have inferiority complexes because they are INFERIOR and cannot make it in the real-world competitive marketplace. If someone wishes to copy one of our copyrighted programs for a friend, by all means go ahead and do so. Just leave our name on it, and if your friend likes it, possibly he will buy the next one and return the favor to you. We could care less, though in a way are flattered that someone would take the trouble to copy it. After all, plagiarism is the HIGHEST form of flattery. You will find numerous singularly SIMILAR versions of Volumes 1 and 2 advertised throughout the land, usually at double the price of our original. Caveat emptor is a good watchword. The ONLY time our back bristles and we call the attorneys is when someone tries to SELL one of our programs for their own. You may make all you want for your friends for FREE, but kindly do not market them or try to rent them for profit. Now, let's get back to computer bulletin board systems which in the next few months will be improved to the point where ANY type of BASIC (any language), or source code, or object code program can be as easily transferred from one place to another, as we describe doing so via amateur radio in the next Chapter.

WHAT DO YOU THINK ABOUT DISSEMINATING COPYRIGHTED PROGRAMS VIA COMPUTER BULLETIN BOARD SYSTEMS TO ANY AND ALL COMERS ? ? ?

Well Gridley, I do not see how you can stop it. Ipso facto, an unenforceable law, if indeed it exists. As such, I believe the bulletin board operators WILL become a significant market in the next few years. Again, IF they start charging the users for the privelege of copying copyrighted programs that the users HAVE NOT PAID FOR, they should be prosecuted UNLESS the copyright owner and computer bulletin operator have reached an accomodation. When you tape a copyrighted song from the radio, you have done nothing wrong. BUT, when you start selling that tape, or renting that tape, or play it through a commercial broadcast station, you're just a plain 'ole thief.

WHAT ABOUT THE ARGUMENT THAT COPYING WILL DRIVE ALL THE GOOD PROGRAMMERS OUT OF THE BUSINESS ? ? ?

Bull roar, Gridley. Just the opposite will happen. The good

ALWAYS drives out the bad whether we are discussing genes and evolution or good and bad programs. That argument is just another CRUTCH for the inferior programmer to bolster his ego.

MINIMUM EQUIPMENT NECESSARY TO ACCESS COMPUTER BULLETIN BOARDS

We have tried some of the kits for building your own RS-232C interface and MODEM and had planned to included them in this Chapter. Our results were marginal at best due to the tolerance and frequency stability of a number of the components that were supplied, especially with the MODEMS. Even with professional grade signal generators and digital frequency counters to align these kits, the results were so disappointing, regarding stability, we decided NOT to cover this aspect of the subject. IF you are a real homebrew enthusiast who does not mind realigning a MODEM's phase lock loop every time the temperature changes, by all means have a go at it. They CAN be built and MADE to work some of the time. For the most part though, they are junk and best left to intrepid experimenters who wish to reinvent the wheel.

The Radio Shack RS-232C adaptor board is well constructed and reliable, IF you purchase the connector brace to reinforce the RS-232C to expansion interface connector on the Model 1 TRS-80. This connector is undoubtedly the WORST design they could have possibly used and will give you nothing but grief till reinforced. The connector brace is a modest $5. ppd. and the best investment you can make for reliability. It is available from: Gunn Industries, 704 Franklin Blvd, Austin, Texas 78751. The Model 3 TRS-80 has corrected this problem.

For a Modem, the Radio Shack Telephone Interface II, #26-1171 at $199 plus the #26-3014 connecting cable at $19.95 is one way to go. This MODEM is manufactured by Novation and is the Model 'CAT' part number 490190 when purchased from Novation dealers for about $30-40 less than the Radio Shack price. It is IDENTICAL to the one with the Radio Shack label. Our first MODEM was damaged in shipment. Novation repaired it free of charge and returned it postpaid in about 10 days. For those who desire a schematic of everything, Novation will ship you the large manufacturing blueprints & schematic for $25. It is an extremely well designed unit that should last a lifetime if treated with modest care; i.e., do not run a truck over it.

PROGRAMMING THE RS-232C ADAPTOR:

Is a journey to the land of frustration if you have the Radio Shack #26-1145 instruction booklet. Photos are upside down and switch numbers are backwards. Hopefully, later versions have been re-written in the English language. Our best advice is to pretend this book was never written and ignore it. Here is how set up the RS-232C adaptor. We call it an adaptor to avoid confusion with the expansion INTERFACE. Four PORTs, numbers 232 through 235, are used by the RS-232C adaptor.

A BRIEF COMMENT ON RS-232C AND WHAT IT IS ALL ABOUT:

The Electronic Industries Association is composed of competing firms, obviously in some aspect of the electronics field. They form committees to develop standards for everything imaginable that would affect the operation of their products, and HOPEFULLY make them more reliable and useful to the end user; THUS making them more saleable. Like motherhood and the flag, this is GOOD. RS-232C is the 3rd revision of RS-232. This standard is entitled: "Interface Between Data Terminal Equipment and Data Communication Equipment Employing Serial Binary Data Interchange," and dated August 1969. It is NOT exactly a NEW standard in any sense of the word.

The Radio Shack Telephone Interface II, is NOT a full RS-232C interface, but relatively close nevertheless. Here's a list of the RS-232C interface pin assignments, their function, and Radio Shack's use or non-use of these pin assignments.

Pin Number	Description	Radio Shack Interface II
1	Protective ground	yes
2	Transmitted data	yes
3	Received data	yes
4	Request to send	not used
5	Clear to send	no - paralleled with 6/8
6	Data set ready	no - paralleled with 5/8
7	Signal gnd (common)	yes
8	Receive sig. detector	yes, paralleled with 5/6
9	Data set testing	yes, +16 vdc
10	Data set testing	yes, -10 vdc
11	Not assigned function	-
12	Secondary sig. detector	not used
13	Secondary clr. to send	yes
14	Secondary xmit data	not used
15	Xmit signal timing	not used
16	Secondary recv. data	yes
17	Recv. signal timing	not used
18	Not assigned function	-
19	Secon. request to send	not used
20	Data terminal ready	not used
21	Signal quality detect	not used
22	Ring indicator	not used
23	Data Signal rate select	not used
24	Xmit element timing	not used
25	Not assigned function	-

If you wish, you may obtain a copy of the RS-232C standard for $5.10 from: EIA, Engineering Dept., 2001 "I" Street, NW, Washington, D.C. 20006.

The only significant item missing from the Radio Shack RS-232C Telephone Interface II is the RING INDICATOR that goes to pin 22, but this may be made by you intrepid experimenters. All you need is a 30 cycle actuated 'RING' detector that causes a solenoid to effect release of the phone 'on the hook' switch.

Having an automatic telephone answering system for the TRS-80 is absurdly simple IF you use the Radio Shack #45-254 low cost telephone answerer ($59.95) as the ring detector AND to close the switch when the calling party is done. More about this aspect in Volume 4. Let's get back to PORTs 232 - 235 and see how the RS-232C adaptor utilizes them.

TAKE NOTE:
For our RS-232C adaptor discussion, bit zero is the least significant bit and bit 7 the most significant bit; i.e., 10000000 binary = 128 decimal, so bit 7 the MSB =1 and bit 0 the LSB = 0. This can be confusing since there is really NO universally agreed upon convention and it can be EITHER way unless specified. The DATA input and output via PORTs 232 - 235 assumes you are using the Radio Shack Telephone II MODEM and Radio Shack RS-232C adaptor board.

PORT 232 - RESET LATCHES AND MODEM STATUS REGISTER:

PORT 232 - OUT232,xx or OUT (232),A (any value for xx or A):
This resets the master data latches for PORTS 233 and 234 allowing you to change Baud rate, parity convention, word length, stop bits, and turn parity ON or OFF.

```
PORT 232       - INP(232) or IN A,(232):
BIT 7 (MSB)    - receive signal detector 0 = ON and 1 = OFF
BIT 6 (NMSB)   - same    as    above       "           "
BIT 5 (NMSB)   - same    as    above       "           "
BIT 4 (NMSB)   - not used by Telephone II MODEM (ring indicator)
BIT 3 (NMSB)   - not used
BIT 2 (NMSB)   - not used
BIT 1 (NMSB)   - received data for UART input
BIT 0 (LSB)    - not used
```

PORT 233 - SET BAUD RATE & READ CONFIGURATION SWITCHES:

```
PORT 233       - OUT233,xx or OUT (233),A:
BIT 7 (MSB)    - 0 for 110 Baud and 0 for 300 Baud
BIT 6 (NMSB)   - 0 for 110 Baud and 1 for 300 Baud
BIT 5 (NMSB)   - 1 for 110 Baud and 0 for 300 Baud
BIT 4 (NMSB)   - 0 for 110 Baud and 1 for 300 Baud
BIT 3 (NMSB)   - 0 for 110 Baud and 0 for 300 Baud
BIT 2 (NMSB)   - 0 for 110 Baud and 1 for 300 Baud
BIT 1 (NMSB)   - 1 for 110 Baud and 0 for 300 Baud
BIT 0 (NMSB)   - 0 for 110 Baud and 1 for 300 Baud

PORT 233       - INP(233) or IN A,(233):
```

We will NOT waste our time with the configuration switch settings as they have NOTHING to do with the operation of the RS-232C adaptor in a PROPERLY written program. Make sure that the COMM/TERM switch is in the TERM position, which is to the REAR of the expansion interface.

PORT 234 - UART CONTROL AND STATUS REGISTERS:

```
PORT 234       - OUT234,xx or OUT (234),A:
BIT 7 (MSB)    - 0 = odd parity and 1 = even  parity  (if  used)
BIT 6 (NMSB)   - 0 = 7 bit word and 1 = 8 bit word
BIT 5 (NMSB)   - 1 = 7 bit word and 1 = 8 bit word
BIT 4 (NMSB)   - 0 = 1 stop bit and 1 = 2 stop bits
BIT 3 (NMSB)   - 0 = parity ON and 1 = OFF
BIT 2 (NMSB)   - 1 always used for all practical purposes
BIT 1 (NMSB)   - 0 always used; NOT connected to Telephone II
BIT 0 (LSB)    - 1 always used: NOT connected to Telephone II

PORT 234       - INP(234) or IN A,(234):
BIT 7 (MSB)    - 1 = receive byte READY in port 235; 0 = NOT
BIT 6 (NMSB)   - 1 = byte transmitted, READY another; 0 = NOT
BIT 5 (NMSB)   - 0 = no overrun error & 1 = ERROR
BIT 4 (NMSB)   - 0 = no framing error & 1 = ERROR
BIT 3 (NMSB)   - 0 = no parity  error & 1 = ERROR
BIT 2 (NMSB)   - not used
BIT 1 (NMSB)   - not used
BIT 0 (NMSB)   - not used
```

PORT 235 - PARALLEL BYTE TRANSMIT AND RECEIVE PORT:

PORT 235 - OUT235,xx or OUT (235),A
This is the port to which we send our parallel byte that is to be transmitted in serial format with start bit, 7 or 8 data bits, parity bit if specified, and 1 or 2 stop bits. The byte is first transferred to the holding register where it is then clocked out a bit at a time by the UART, at whatever Baud rate we specified earlier. The data bits are transmitted backwards; i.e., bit 0 (LSB) first and bit 7 (MSB) last. After ALL the bits are transmitted, the RS-232C adaptor then outputs a 1 = 'I am ready for another byte' that is THEN loaded into bit 6 which may be tested via port 234. We have to make this test before outputting another byte via port 235 as there is only 1 transmitter holding register in this system. BASIC is so slow that it is unnecessary to check the status of bit 6 when transmitting with a BASIC program at 110 Baud. At 300 Baud, BASIC is so slow as to be impractical, so we will use assembly language for our bulletin board program.

PORT 235 - INP(235) or IN A,(235)
Is the port to which the received 7 or 8 data bit serial byte now converted to parallel is sent AFTER the conversion is completed by the UART. The RS-232C in it wisdom then outputs a 1 = 'serial byte converted and READY for delivery' that is THEN loaded into bit 7 which may be tested via port 234. We may test it with IN A,(234) - BIT 7,A - RET Z in assembler or IF INP(234) <128 go back and look again, in BASIC. The reason for testing it is that we do not want an uncompleted byte loaded into port 235, since our SPEEDY assembly language program is many, many orders of magnitude FASTER than the Baud rate we are using. Even BASIC's low speed requires that this check, as listed above, be made BEFORE using the byte.

YOU SURE EXPECT US TO ABSORB A LOT, AWFULLY FAST ! ! !

Not really, Gridley. Put it under your pillow tonight and maybe you will pick it up via osmosis, like you did in Volume 1.

COMPUTER BULLETIN BOARD CUSTOMS AND PRACTICES:

Vary somewhat between ABBS, CBBS, and Forum-80 systems. The best way to get started IF you do not have a friend nearby who is already an expert (all who use them are EXPERTS), then read up on the subject in back issues of Kilobaud and 80 Microcomputing magazines. Yet another way to go is to write Forum-80 (tm) Headquarters at 7600 East 48th Terrace, Kansas City, Missouri 64129 and enclose a self-addressed, TWO STAMPS PLEASE, envelope for a freebie "Forum-80 3.0 Users Guide." The price is obviously right and this little 20 page booklet of interesting information will be of considerable help to you in getting started.

MOST bulletin board systems are designed to accept the more popular microcomputers such TRS-80 models 1 and 2, Apple II, Pet, Heathkit H-18, Soroc, etc. After signing on a given bulletin board with your name and location for the FIRST time, the most important item is to tell it what type of configuration (terminal) you are using since the control code protocols between the various models of microcomputers are VERY different. You do not want a CLS everytime you receive a carriage return or vice-versa. After you have logged onto the bulletin board with your name and location you want to get into the COMMAND mode as quickly as possible. Just answer any questions logically and follow instructions. When you receive the querry "COMMAND:" answer with a 'C' and ENTER. This should place you in the configuration change mode and the response should be: LOADING FILE (C) CONFIGURATION CHANGES folowed by: (C) SUBCMD. Press the question mark key and then ENTER. The video display (for Forum-80) will be as follows:

```
S = SCREEN CLEAR               C = CURSOR CONTROL
N = NULLS                      L = LINEFEED SWITCH
I = INSTRUCTIONS               T = TERMINAL LISTING
V = VERIFY CONFIGURATION       A = ABORT & RET TO COMMAND MODE

(C) SUBCMD:
```

Since you most likely are using a Model 1 TRS-80, press 'T' and then ENTER. You will be given a list of terminals on file which will include your Model 1. Then key in MOD1 and ENTER.

IF you use this particular bulletin board frequently, hopefully the system will keep your configuration in its files so that when you log on, it will automatically recognize your name and location and thusly, your configuration too. Most all bulletin board programs are designed to work with most anything from a truly 'dumb' terminal on up through the more sophisticated TRS-80 Model II, and most everything in between. As such, configuration is an absolute necessity.

MY RS-232C ADAPTOR HAS THE REINFORCING BRACKET AND STILL IS OCCASIONALLY INTERMITTENT, PLUS THE CARBON GRANULES IN MY TELEPHONE II MODEM'S MICROPHONE OCCASIONALLY 'PACK-UP'! ! ! IS THERE ANY ALTERNATIVE ? ? ?

Sure there is, Gridley. Our old friend, Donald Stoner-W6TNS, has designed a modem for direct connection to your phone lines at one end, and either the 'barefoot' connector on the TRS-80 keyboard or the screen printer port connector on the E/I at the other end. Using Don's new modem eliminates the RS-232C adaptor board (no connector problem) and also eliminates the acoustic modem, the Telephone II. It is called the 'Micro-Connection for TRS-80' and sells for $249.

It is manufactured by Don's firm, The Peripheral People, P.O. Box 524 - Dept. W4UCH, Mercer Island, Washington 98040. Mention the 'Disassembled Handbook For TRS-80,' and they will pay the shipping. We have spoken with users, and all give it first rate reports.

Now that we have solved your hardware problems, Gridley, let's go back to the software aspects of making the Radio Shack RS-232C adaptor and MODEM work with most any variety of bulletin board. We will make it as painless and simple as possible. Once you start using bulletin board systems, you will find a 'zillion' ways to improve this fundamental program, but for now, KISS = Keep It Simple Simon.

Here are our objectives for the fundamental bulletin board program:

1. Transmit and receive at the standard 300 Baud rate used by all KNOWN bulletin board systems today.

2. Allow the user to select 7 or 8 bit mode on intialization and after the program is running to change modes when desired. Gridley MAY wish to send graphics/pictures to his friends AND/OR he MAY wish to swap programs with his his buddies. Remember Gridley, programs require 8 bits.

3. Work FULL DUPLEX; i.e., the bulletin board echoes back the transmitted byte allowing us to visually check the accuracy of the received data at the bulletin board end.

4. Further FULL DUPLEX; i.e., the program automatically scans incoming signals and IF not present then scans the keyboard so that NO manual transmit/receive switching is required.

THAT'S A PRETTY TALL ORDER, PROFESSOR. JUST HOW YOU GONNA DO IT IN LESS THAN A FEW THOUSAND BYTES ? ? ?

Come on, Gridley. You are acting gun-shy again. Actually the program is only a 100+ bytes long and fits on a single page. Do not let those $50 & $100 programs scare you into buying one till you try writing one yourself. Welcome to simplesville.

```
; TRS-80 BULLETIN BOARD DEMONSTRATION PROGRAM - BB1 & 2
;
W6TNS   EQU     32000           ;= 7D00H FOR 16 FINGERS
        ORG     W6TNS           ;START THE PROGRAM HERE
        OUT     (232),A         ;MASTER RESET LATCH
        CALL    01C9H           ;CLS ROM SUBROUTINE
        LD      SP,32760        ;MOVE STACK HERE IN MEM
        LD      A,55H           ;55H = 300 BAUD RATE
        OUT     (233),A         ;SET BAUD VIA PORT 233
        LD      HL,MESS1        ;STRING MEM ADDRESS
        CALL    28A7H           ;DISPLAY STRING ROUTINE
MESS1   DEFM    'A = SEVEN BITS & B = EIGHT BITS ? '
        DEFB    00              ;END OF MESSAGE DELIMITER
        CALL    049H            ;AWAIT KEYBOARD INPUT 'A'
        CALL    032AH           ;DISPLAY INPUT ON VIDEO
        CP      66              ;B=66 SUBTRACT FROM 'A'
        JP      Z,LOAD          ;GOTO 8 BIT LOAD IF ZERO
        LD      A,13            ;13 = CONTROL SKIP A LINE
        CALL    033H            ;DO IT ON VIDEO
        LD      A,165           ;7 BITS, 1 STOP, PAR 'ON'
        OUT     (234),A         ;INITIALIZE VIA PORT 234
LOOK    LD      A,(14400)       ;= KEYBOARD CLEAR KEY ROW
        CP      2               ;2 = CLEAR KEY PRESSED
        JP      Z,W6TNS         ;CHANGE BITS TO 7 OR 8
        LD      A,14            ;CONTROL 14 = CURSOR 'ON'
        CALL    033H            ;DISPLAY CURSOR ON VIDEO
        CALL    RECV            ;GOSUB RECEIVE SUBROUTINE
        CALL    XMIT1           ;GOSUB TRANSMIT SUBRUTINE
        JP      LOOK            ;TAKE ANOTHER LOOK AGAIN
RECV    IN      A,(234)         ;RECEIVED BYTE CONVERTED?
        BIT     7,A             ;TEST BIT 7 SET 'Z'
        RET     Z               ;GOTO XMIT 'IF' NOT READY
        IN      A,(235)         ;INCOMING CONVERTED BYTE
        JP      Z,RECV          ;IF ZERO, LOOK AGAIN
        CALL    033H            ;IF NOT ZERO, DISPLAY IT
        JP      RECV            ;LOOK AGAIN AT INCOMING
XMIT1   CALL    02BH            ;ANY KEYBOARD OUTPUT ?
        OR      A               ;SET 'Z' FLAG IF 'NOT'
        RET     Z               ;GOTO LOOK AGAIN IF ZERO
        PUSH    AF              ;SAVE AF TO TEST UART RDY
XMIT2   IN      A,(234)         ;HOLDING REGISTER EMPTY?
        BIT     6,A             ;TEST BIT 6 SET 'Z' FLAG
        JP      Z,XMIT2         ;NOT EMPTY? TEST AGAIN
        POP     AF              ;EMPTY! RESTORE 'A' REG
        OUT     (235),A         ;BYTE TO HOLDING REGISTER
        RET                     ;RETURN TO 'LOOK' AGAIN
LOAD    LD      A,13            ;13 = CONTROL SKIP A LINE
        CALL    033H            ;DO IT ON VIDEO
        LD      A,229           ;8 BITS, 1 STOP, PAR 'ON'
        OUT     (234),A         ;INITIALIZE VIA PORT 234
        JP      LOOK            ;GOTO LOOK AGAIN
        END     W6TNS           ;EL FIN = EL BEGUINE
```

- Source Code -

```
7D00           00120 W6TNS   EQU     32000
7D00           00130         ORG     W6TNS
7D00 D3E8      00140         OUT     (232),A
7D02 CDC901    00150         CALL    01C9H
7D05 31F87F    00160         LD      SP,32760
7D08 3E55      00170         LD      A,55H
7D0A D3E9      00180         OUT     (233),A
7D0C 21127D    00190         LD      HL,MESS1
7D0F CDA728    00200         CALL    28A7H
7D12 41        00210 MESS1   DEFM    'A = SEVEN BITS & B = EIGHT BITS ?'
7D34 00        00220         DEFB    00
7D35 CD4900    00230         CALL    049H
7D38 CD2A03    00240         CALL    032AH
7D3B FE42      00250         CP      66
7D3D CA807D    00260         JP      Z,LOAD
7D40 3E0D      00270         LD      A,13
7D42 CD3300    00280         CALL    033H
7D45 3EA5      00290         LD      A,165
7D47 D3EA      00300         OUT     (234),A
7D49 3A4038    00310 LOOK    LD      A,(14400)
7D4C FE02      00320         CP      2
7D4E CA007D    00330         JP      Z,W6TNS
7D51 3E0E      00340         LD      A,14
7D53 CD3300    00350         CALL    033H
7D56 CD5F7D    00360         CALL    RECV
7D59 CD6F7D    00370         CALL    XMIT1
7D5C C3497D    00380         JP      LOOK
7D5F DBEA      00390 RECV    IN      A,(234)
7D61 CB7F      00400         BIT     7,A
7D63 C8        00410         RET     Z
7D64 DBEB      00420         IN      A,(235)
7D66 CA5F7D    00430         JP      Z,RECV
7D69 CD3300    00440         CALL    033H
7D6C C35F7D    00450         JP      RECV
7D6F CD2B00    00460 XMIT1   CALL    02BH
7D72 B7        00470         OR      A
7D73 C8        00480         RET     Z
7D74 F5        00490         PUSH    AF
7D75 DBEA      00500 XMIT2   IN      A,(234)
7D77 CB77      00510         BIT     6,A
7D79 CA757D    00520         JP      Z,XMIT2
7D7C F1        00530         POP     AF
7D7D D3EB      00540         OUT     (235),A
7D7F C9        00550         RET
7D80 3E0D      00560 LOAD    LD      A,13
7D82 CD3300    00570         CALL    033H
7D85 3EE5      00580         LD      A,229
7D87 D3EA      00590         OUT     (234),A
7D89 C3497D    00600         JP      LOOK
7D00           00610         END     W6TNS
00000 TOTAL ERRORS
```

\- Object Code -

HOW COME YOU USE JP'S INSTEAD OF JR'S IN THIS PROGRAM ? ? ?

Another good question, Gridley. Remember, back in Volume 2 we found that JR's took one byte less MEM than JP's?

YES. THAT'S WHY I ASKED THE QUESTION.

Very good. But, you seem to have forgotten that JR's take 20 percent MORE time for the Z-80 to process than JP's. Memory is so cheap today, we have gotten into the habit of using JP's for just about everything. IF you started running this program at Baud rates in the vicinity of 19,200 you might find that program execution time became a limiting factor. Just to be on the safe side, let's try to use JP's from now on, UNLESS memory space becomes a problem.

A QUICK RUN-THROUGH THE BULLETIN BOARD DEMONSTRATION PROGRAM: (we will skip program lines that are obvious to Gridley)

Line 140:
Triggers the RS-232C master reset latch via OUT port 232. ANY value for the 'A' register is ok as it makes no difference what the value may be.

Line 160:
Moves the stack pointer's MEM location up out of BASIC territory where it cannot get into trouble. It is NOT necessary in this particular program as it stands, but if you expand it, as we expect you to do, it is good programming practice and a good habit to become accustomed to using on a regular basis as fouling it up can spoil your whole day.

Line 170:
OUT port (233),A with A = 55H loads the RS-232C baud rate register with 01010101 binary = 300 Baud, which is the standard rate for most all bulletin board systems.

I REMEMBER THAT WE USED 22H = 00100010 BINARY FOR 110 BAUD. CAN WE SET UP THE BAUD RATE REGISTER TO SAY TRANSMIT AT 300 BAUD AND RECEIVE AT 110 BAUD, OR VICE VERSA ? ? ?

Sure we can, Gridley. Bits 7 through 4 determine the transmit Baud rate and bits 3 through 0 the receive Baud rate. To transmit at 300 Baud = 0101 nibble and receive at 110 Baud = 0010 nibble equals a total binary number of 01010010 = 52H and 82 decimal. For your vice versa, we reverse the nibbles to yield 00100101 = 25H and 37 decimal to transmit at 110 Baud and receive at 300 Baud. This is not an uncommmon practice in radio teletype (tm), but NEVER used with bulletin boards.

Line 260:
Sends us off to line 560 IF we selected 8 bit transmission and reception. IF we selected 7 bits, then it falls through to lines 270 to 300 where it effects a carriage return and then sets up 7 bits, 1 stop, and parity ON via port 234. Our mini-

program does NOT check for any parity error via bit 3 of the UART status register (port 234) for incoming bytes, though most of the bulletin board programs at their end do. It is easy to add if you wish.

Line 360 and 370:
Are the real work horses of this program alternately switching back and forth between RECV and XMIT1. The comments are mostly self-explanatory. The most important item to note is that RECV has priority over XMIT; i.e., we cannot transmit a byte as long the UART status register (port 234) bit 7 indicates that a fully converted (serial to parallel) byte is available and set up to read from port 235. It would really require TWO Z-80 microprocessors with their own memory or FIFOs to allow us BOTH to transmit at EXACTLY the same time and make any sense out of the simultaneous transmissions.

YOU FORGOT TO SHOW US HOW TO TRANSMIT, RECEIVE, AND STORE SOME PROGRAMS TO & FROM COMPUTER BULLETIN BOARDS ! ! !

Be patient, Gridley. We have got to save something worthwhile for the next Chapter. Especially for those readers who have no interest in amateur radio. By placing the program movement and storage section in Chapter 10, at least they will read the part of the Chapter that interests them and MAYBE even get bitten by the amateur radio bug. Really, there are worse hobbies.

MY MOTHER DOESN'T THINK SO ! ! !

Just wait till you discover girls NOT wearing jeans, Gridley.

CONCLUSION:

This program may be checked by placing a sheet of folded paper over your MODEM; right switch in TEST and left in Originate. We have covered ONLY the briefest of outlines of computer bulletin board systems in this Chapter. The subject truly deserves an entire book to do it justice. Probably the most fascinating aspect of this subject would be writing an assembly language program that would allow the reader to set up his/her own bulletin board central station with automatic telephone answering and all the goodies that go with it when running 3 or 4 disks with a Model 1 (probably 2 disks would handle it with a Model 3).

To do the job properly would require learning the protocols of at least the Model 2, Model 3, Apple II and Pet, unless you wished to confine it only to original vintage TRS-80s.

IF you really want to get into the bulletin board aspects of our hobby VERY deeply, the Forum 80 (version 3.0) central station program is available for ONLY $150 from Forum 80 Headquarters. Their address is listed earlier in this Chapter. The real FUN & GAMES will be using 2 meter amateur repeaters.

- CHAPTER 10 -

RADIO TELETYPE FROM 'A' TO 'Z'

INTRODUCTION:

The title for this Chapter is possibly a bit too ambitious in that it infers we will cover EVERYTHING concerning radio teletype. Possibly a better title would be, 'MOST EVERYTHING ABOUT RADIO TELETYPE FOR THE TRS-80 USER.' What we do not need is a lengthy exposition on Model 33 electromechanical teletype machines and Baudot teletype code, in this day and age of computer generated and decoded ASCII RTTY code. Even the venerable American Radio Relay League (ARRL) station, W1AW, has been transmitting ASCII as well as the old Baudot RTTY code since July 1980. As such, we will treat Baudot as an interesting 'historical' aspect of RTTY operation and even review its use with the excellent Macrotronics M-80 system for the TRS-80, but MOST of this Chapter will concentrate on ASCII transmission and reception of RTTY on the amateur radio bands.

A BIT OF HISTORICAL PERSPECTIVE:

Circa 1901, an extremely bright U. S. Army Signal Corps. Captain by the name of Squires invented a multiplex system for radio telegraph transmission and reception; i.e., multiple circuits using a single wire and ground. IT was the technical breakthrough that led to the invention and development of the first American teletypewriter during 1905 - 1906 by the famed inventor, Charles Krum. Squires later became the Commanding General of the U. S. Army Signal Corps. during World War I, and was largely responsible for foistering the development of the American electronics industry. If any one man deserves the credit for America's predominant position in electronics today, General Squires name would lead the list. He encouraged development of the teletypewriter, vacuum tube, reliable ground based military communications systems, air to air and air to ground communications systems, but to name a few of his contributions.

Charles Krum's financial backing for his new teletype machine was through the foresight of Jay Morton, (Morton Salt Company) an entreprenour of the 'Henry Ford' variety who had the guts to see the company, 'MORKRUM' through the difficult growing pains that most all new firms experience when introducing a revolutionary new product into the marketplace. World War I AND the adoption of the Morkrum teletypewriter by the Associated Press news service in 1915, put the firm into the worldwide leading position in its field.

In 1925, Mokrum and its only significant U.S. competitor, the Kleinschmidt Company were merged as anti-trust actions by the Department of Justice were then rather few and far between.

Bell Telephone, knowing a good thing when the price WAS right, purchased the combined firms in the depression year of 1930 and named them the Teletype Corporation. TELETYPE is a registered trademark of the Western Electric Corporation which is Bell Tel's manufacturing subsidiary. To avoid improperly using this trademark, we will henceforth use RTTY and/or TTY when we mean either radioteletype OR teletype.

The argument as to WHO invented the Baudot code still lingers on. Whether it was Monsieur Baudot, Packard, or Bell is of no importance to us as it has gone, is going the way of the buggy whip industry. The TTY machine in its time and place was a wonderful contraption that sure beat green-eye-shaded telegraphers hunched over manual typewriters to copy the message being transmitted. The reliability of these clanking-whale-oil-smelling TTY machines was remarkably good considering the number of moving parts just waiting to fail. We like to compare their reliability to that of the helicopter. One knows it is going to fail sometime. You just do not know WHEN.

At the end of World War II there were zillions of surplus TTY machines available for practically nothing. The ham radio/ amateur radio market was really 'ripe' and the 'right' place for these freebie goodies. When ubiquitous WAYNE GREEN's submarine finally surfaced and was informed that the "war was over," Wayne started a newsletter promoting these freebie TTY goodies for the amateur radio fraternity. "What giant oaks do little acorns grow." Wayne went on to become the Editor of CQ MAGAZINE. We have a CQ Magazine VHF/UHF Achievement Award hanging on the wall signed by Wayne and and our late friend, Sam Harris. Wayne then continued onwards and upwards by founding 73 MAGAZINE, BYTE MAGAZINE, KILOBAUD-MICROCOMPUTING MAGAZINE, and 80 MICROCOMPUTING MAGAZINE, while simultaneously gadflying ARRL to keep them on the straight and narrow path of righteouness.

Meantime, those zillions of surplus whirring-clanking-growling TTY machines found their way into the hamshacks of ham radio operators. A TTY fraternity within a fraternity of ham radio operators was spawned dedicated unswervingly to operating AND repairing these marvelous/wondrous devices that MARS (Military Amateur Radio System) gave out for 'free' to anyone that would pick them up. Seriously, they had changed little from the 1905 Morkrum TTY machines. Their is no question as to the 'addictive nature' of this hobby within a hobby. One of our friends in Leesburg, Virginia, had the ENTIRE downstairs portion of his otherwise lovely home covered with wall to wall Model 15 TTY consoles. One for each amateur band from 75 meters THROUGH 2 meters. After his wife left him, keeping these wondrous clunkers clunking occupied ALL his free waking time when not acting as engineer at the local radio station.

Little changed till about 1975 when two important events transpired that would markedly change this otherwise weird scene: 1. A series of articles in QST Magazine outlining 'how to build a solid-state RTTY transmitting and receiving system.

2. Hal Communications Corp. in Urbana, Illinois decided to expand their Morse code keyer business and introduced one of the first solid-state 'factory built' RTTY systems for amateur radio operators.

At last, one could vicariously share the RTTY buff's thrill of sending and receiving the printed word WITHOUT having to lose their wife and could do so silently without the smell of whale oil pervading their clothes and premises. For the real 'build it yourself buff,' option #1 was the way to go. For those who preferred 'store bought' gear, option #2 was the alternative. Numerous early MITS and ALTAIR microcomputer hobbyists wrote Baudot code conversion programs to use with their 8008 or 8080 based micros. Even monolithic Texas Instruments and Motorola came out with Baudot to ASCII and vice versa solid-state converter chips since Baudot was the only TTY code authorized for ham radio use by the Federal Communications Commission.

The stage was now set for the main attraction. The BIG Saturday Night Movie, a double feature should have been entitled: "Tandy's Folly Or How The TRS-80 Captured The World Marketplace For Microcomputers," and "The F.C.C. Awoke In 1980 And Authorized ASCII Amateur Radio Transmission." BOTH movies are true-real-life stories and what this Chapter is all about. So much for past history and water over the dam. "What's done is done and all behind us," the Chinese say. Let's see what we will try to cover henceforth.

We will divide this Chapter into three segments:

FIRST: Assembling a 1/2-duplex VHF dual-band ASCII data link using the RS-232 adaptor and CAT modem that utilize 6 meter and 2 meter amateur transceivers.

SECOND: A brief look at the Macrotronics M-80 Ham Interface using either Morse or Baudot for transmission and reception on the amateur radio HF bands.

THIRD: A radio teletype (tm) receiving program requiring only 3 simple lines that will copy the daily W1AW RTTY bulletins transmitted in ASCII + a surprise.

ASSEMBLING A HALF DUPLEX 6 & 2 METER BAND ASCII DATA LINK:

What did he say ? ? ?

Let's take it word by word, Gridley. Simplex communication is the way normal HF (high frequency 1.8 to 29.7 Megahertz) amateur radio signals are transmitted and received. The clue to simplex is that BOTH stations use a common frequency. It is obvious that ONLY one station can transmit at a time. When using phone, the normal words that signify a given station's end of transmission are "go ahead." In Morse code, the letter 'K' means the same thing. The transmitting station may omit the "go ahead" by saying "W1ABC this is W4DEF" which implies a "go ahead" to the other station and identifies them both.

Duplex communication is much like YOU on the telephone with your mother, Gridley. By that I mean that you both can talk at the same time. Whether either party receives any useful information when both are talking simultaneously is a matter of conjecture we will not address here, but with certain types of computers and program formats, especially those employing dual microprocessors with store and forward capability, DUPLEX is a real time saving feature. Even with our single micro-processor TRS-80, duplex operation offers considerable advantages, the most important being NOT having to switch back and forth from transmit to receive and vice versa, as we did with simplex. Depending on the 'convention' adopted by the full-duplex users, the signal MAY be echoed back by one or both of the receiving parties thus allowing a parity check at each transmit end of the link to make sure the correct signal was received at the other end of the link. IF it was not correctly received, then it MAY be re-transmitted IF desired. Most full-duplex systems display ONLY the echoed back signal at the transmit source which obviously requires TWO frequencies UNLESS the users are willing to switch back and forth from transmit to receive for EACH character which is way to costly in TIME.

Half-duplex is nothing more than a convention adopted by a number of users that signifies that the character transmitted is simultaneously displayed on the transmit video display and NO echo back and/or parity check is accomplished.

The 6 and 2 meter bands are simply those frequencies from 50 to 54 MHz and 144 to 148 MHz, respectively, that the FCC and ITU have allocated for amateur radio use in our portion of the western hemisphere since World War II, including both the US and Canada. We qualify this statement since in most all of Europe the 6 meter band is allocated to television channels.

ASCII means just what it says. We will be transmitting and receiving serial ASCII characters instead of Baudot or Morse code using FSK (frequency shift keying) modulation.

Data link infers that we will be transmitting and/or receiving data rather than voice communication, though at times SOME data links utilize voice communications too. A better term would be digital data link which would usually be utilized solely for data communications, but just as before, could handle digitized voice communications too.

THE FREQUENCY PLAN - USING THE 6 AND 2 METER AMATEUR BANDS:

WHY ARE WE USING 6 & 2 METERS ? ? ?

Well Gridley, for a number of hopefully good reasons. Here they are:

1. To operate on these VHF frequencies, and higher, all that is required is an FCC Technician Class license which is easy

to obtain. All that is required is that the applicant pass a 5 words per minute Morse code test (the same as for the Novice Class license) and the General Class written examination which most TRS-80 users could master by studying the ARRL examination guide in the evening for a week or so. For newcomers to amateur radio we recommend they obtain from the American Radio Relay League, 225 Main Street, Newington, Conn. 06111:

- TUNE IN THE WORLD WITH HAM RADIO $7.00 POSTPAID
  (1 hour Morse code practice cassette + novice theory book)

- TECHNICIAN/GENERAL Q & A BOOK $2.50 POSTPAID
  (study guide & reference book for FCC written examination)

2. The 6 meter and 2 meter bands offer excellent line of sight transmission paths with very LOW power required. By LOW power we mean that a watt or two into a simple 3 element beam antenna will allow you to reliably work most ANY line of sight distance imaginable ON EARTH.

3. We also chose the 6 and 2 meter bands for the reason that relatively INEXPENSIVE transmitting and receiving converter kits are available if you are already are a radio amateur. IF NOT, then one possible way to go would be to start on the 6 and 2 meter bands using the OUTSTANDING "ICOM" transceivers for these bands that offer am, fm, single sideband, and CW transmission and reception capabilities at modest cost.

* * * ED. NOTE * * *
The author knows whereof he speaks regarding the VHF/UHF and microwave amateur bands as he held the U.S. record for number of countries worked on the 6 meter band through the last 2 sunspot cycles. During that time he had worked 36 countries on 6 meters and he holds the I.A.R.U. Award issued by the ARRL in 1960 of "Worked All Continents" WITH THE 6 METER ENDORSEMENT He is also author of the "Gunnplexer Cookbook - A Microwave (10 GHz) Primer," now being published (1981) by Communications Technology Inc., Greenville, N.H. 03048.

Both our frequency plan and the equipment required to implement it are simplicity personified. One station will transmit on 6 meters and receive on 2 meters while the other station will transmit on 2 meters and receive on 6 meters. For purposes of illustration we will assume that station A transmits on 51.000 MHz and station B transmits on 145.000 MHz. Actually, ANY clear channel in the local area could be used on these bands that WOULD NOT interfere with a local REPEATER station. IF it did interfere, a definite NO-NO, you probably would have a few dozen extremely irritated ham operators camping on your doorstep until you moved off the repeater frequency. This is no problem though. Just listen on the two frequencies you plan to use BEFORE you decide on the transmitter frequency for each transmitter AND ask some local hams, who are invariably the most HELPFUL types you will ever meet, for the frequencies of local repeater stations on the 6 and 2 meter bands. Should you live in a large metropolian area

where there seems to be 'wall to wall' 2 meter repeaters EVERYWHERE, just pick a frequency of approximately plus or minus 30 kilohertz above or below one of the local repeaters and there should be no problem as the frequency spacing for most all 2 meter repeaters in the U.S. is 60 kilohertz. Their should be no interference problem on the 6 meter band. Also, remember that technician class licensees may use ANY frequency between 50.100 and 54.000 MHz on the 6 meter band and 145.000 and 148.000 MHz on the 2 meter band. Courtesy to DX (long distance) enthusiasts dictates that you avoid the fequencies close to 50.100 MHz UNLESS you are attempting to work DX yourself.

The system we are going to discuss in this section utilizes ONLY 200 cycle frequency shift keying and should NOT interfere with ANYONE if you choose your transmitting frequencies carefully AFTER checking with local amateur radio clubs and/or amateurs familiar with repeater operations in your vicinity.

HOW DOES A SINGLE SIDEBAND (SSB) 6 OR 2 METER TRANSMITTER GENERATE FREQUENCY SHIFT KEYING (FSK) ? ? ?

Quite simply, Gridley. Let's assume our TRS-80 has the RS-232 adaptor installed and we have either the Radio Shack or Novation CAT MODEM plugged into the RS-232 card. The are both identical MODEMS though one has the Radio Shack label and the other the Novation CAT label.

The CAT modem has the FULL/TEST/HALF switch in the TEST position and the ORIGINATE/ OFF/ANSWER switch is in the ORIGINATE position. As such, the MODEM will output a Mark = 2225 cyles and Space = 2025 cycles audio tone from its speaker whenever we activate the program listed in Figure 10-X, AFTER we press a key on the keyboard. Figure 10-1 illustrates the audio signal that the '@' sign = ASCII 64 = 1000000 binary would output with 1 start bit, 7 data bits, and 1 stop bit. Note that the LSB is sent first and the MSB is sent last which is 'sort of' backwards, but that's life

```
BIT              S  0  1  2  3  4  5  6  S
                 T                       T
                 A                       O
                 R                       P
                 T  0  1  2  3  4  5  6  :
                    0  0  0  0  0  0  1          = '@'
```

MARK = 2225

SPACE = 2025

Note that the start bit is ALWAYS a space = 2025 cycle signal and that the '1' = 2225 cycles AS WELL AS the 'stop' bit.

Figure 10 - 1

WHEN ARE YOU GONNA ANSWER MY QUESTION ABOUT FSK ON SSB ? ? ?

Be patient, Gridley. We're almost there. Now, for just a bit of theory with a practical example. If we take a look at the spectrum of an amplitude modulated (AM) radio signal at let's say 51.000 MHz that is modulated with a pure tone of 2225 cycles it would look like Figure 10-2 on a spectrum analyzer/ panadaptor. For the sake of simplicity we will ignore the harmonics.

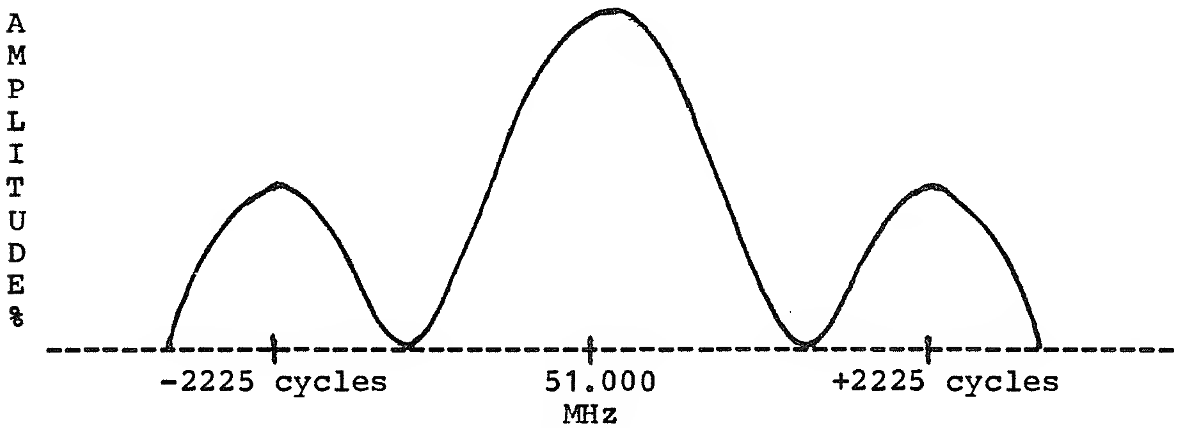


Figure 10-2

Here we can see that 50 percent of the power is in the carrier at 51.000 MHz and the other 50 percent of the radio frequency power is in the 2 sidebands that are 2250 cycles above and below the carrier; i.e., each sideband contains 25 percent of the total radio frequency power. IF we were to filter out or phase out the carrier and forinstance the lower sideband we would ONLY have a signal at 51.000 MHz +2225 cycles = 51.002225 MHz remaining. The is the UPPER SIDEBAND of our original signal. IF we changed the modulating frequency to a Space = 2025 cycles, then the output on USB (upper sideband) would be 51.002025 MHz. By feeding the USB signal to an appropriate linear amplifier we may amplify the signal to any power level desired AND THE OUTPUT WOULD BE PURE FREQUENCY SHIFT KEYING, no more no less. Since our SSB transmitter offers us the facility of either upper OR lower sideband output we may effectively REVERSE the Mark or Space tones as desired. At the receive end, we may do the same thing by switching our SSB receiver to either USB or LSB (lower sideband). This last point about reversing Mark and Space tones is important as our CAT modem and RS-232 adaptor insist that the Mark = 2225 cycles and the Space = 2025 cycles. IF our received signal is INVERTED (vice versa) then GIGO = garbage in garbage out. To correct it, all we need do is switch sidebands on the receiving end of the circuit.

One page is certainly NOT going to explain ALL the aspects of single sideband transmission and reception. We suggest you obtain a copy of the "1981 Radio Amateur's Handbook" which thoroughly covers the subject, IF you wish to dig deeper. It is available from: ARRL, 225 Main Street, Newington, Conn. 06111 for $10. postpaid and is the bargain of a lifetime.

THERE'S GOT TO BE A FLY IN THE OINTMENT......SOMEWHERE ? ? ?
YOU MAKE IT SOUND SO EASY I DON'T BELIEVE IT ! ! !

As usual Gridley, you are right again. The problem is that of radio frequency interference (RFI) generated by our good old TRS-80 itself. Let's take a brief look at the problem and the ways and means we are going to suggest that you solve it.

RADIO FREQUENCY INTERFERENCE GENERATED BY THE TRS-80:

Every little digital gate in the TRS-80 plus the nominal 10.6445 megahertz crystal oscillator that is divided down to 1.774 MHz for the Z-80 clock is in reality a radio frequency generator of the FIRST water. IF you do not believe it, take a small transistor am radio and place it next to the keyboard while you list or run any variety of program. What does it sound like, Gridley?

IT SOUNDS LIKE MACNAMARA'S BAND HAS BEEN DRINKING SOME OF THEIR OWN 'IRISH' AGAIN. IT SOUNDS AWFUL ! ! !

You are right-on, Gridley. It does indeed sound awful. Every little digital gate in the TRS-80 thinks its the TITANIC's spark coil transmitter and send its own SOS message every time it is turned 'on' or 'off.' With this occuring thousands and millions of times per second the HARMONICS generated by these square waves are easily measured up to and beyond 100 to 200 megahertz in frequency. The amplitude of these harmonics that are radiated by the TRS-80 are dependent upon a number of factors. The most important ones are the degree of shielding each active component has PLUS the length and layout of the interconnecting cables that radiate the harmonics too.

SHIELDING ? ? MY TRS-80 IS IN A UNSHIELDED PLASTIC CABINENT! !

Right again, Gridley. Most ALL microcomputers that were designed circa 1976/77 or thereabouts had to consider the cost effectiveness of shielding the whole shebang and its resulting addition to the end user price, or NOT shielding it at all as there were no F.C.C. rules or regulations at that time that required it. The marketplace shows that the three leading microcomputer manufacturers, Radio Shack, Apple, and Commodore were RIGHT in their decision not to shield these initial models as the higher introductory price would have severely limited the market. BUT, there's that big BUT again, the laws of nature have no respect for the marketplace, so most every microcomputer sold through December 31, 1980 was a truly horrendous radio frequency noise generator. In television fringe areas one could not have noise-free TV and the micro-computer running at the same time. The user had to choose one or the other. In 1979, the F.C.C. made the choice for the heretofore laissez-faire, free-trading manufacturers when it (the FCC) decreed that ALL computers manufacturered after July 1, 1980 would be RFI free and clean as knights in shining armour thereafter; no 'rust' allowed to show anywhere.

WHAT DO YOU MEAN ABOUT 'RUST' SHOWING ANYWHERE ? ? ? I THOUGHT THE DATE FOR THE NEW RFI FREE COMPUTERS WAS JANUARY 1, 1981 ?

1. Only a figure of speech, Gridley. What we meant was that that the manuufacturers had until July 1, 1980 to 'clean-up their act' regarding radio frequency interference.

2. You are right again, Gridley. The microcomputer manufacturers obtained a 'stay of execution' till January 1, 1981 by pleading 'not enough time' to reasonably and ECONOMICALLY redesign their microcomputers. The FCC did indeed grant their wish for the time extension. Radio Shack's answer to the FCC requirement was the Model 3 TRS-80 annouunced in early summer 1980. That is what Chapter 1 of Volume 4 of the 'Disassembled Handbook For TRS-80' is ALL about; i.e., the minor differences in the ROM, plus a complete section comparing the radio frequency interference levels of the 2 models from 10 kHz up through 450 MHz. For this Volume though, we will stick to the model 1 TRS-80 and its problems with emphasis on how to minimize them by working AROUND them rather than attempting to CURE them.

WHY DON'T YOU TELL US A SIMPLE AND EFFECTIVE MEANS OF ELIMINATING MODEL 1 RADIO FREQUENCY INTERFERENCE ? ? ?

We would if we could, Gridley. Sad to say, there is no truly simple approach to the RFI problem. The laws of Mother Nature take precedence here as there is NO EASY WAY to reduce RFI to an acceptable level, muchless a SIMPLE one. Without going into the theory and practice of de-RFIing a TRS-80 let us only comment that without an awfully lot of hard work and extra shielding that it is for all intents and purposes impossible to de-RFI a TRS-80 with the expansion interface, a few disk drives, and a couple of cassettes. By the time you had it fully shielded, you would be in virtual radiation proof 'screen room' with your own independent power source. To talk to the outside world would require adequate filtering of each and every lead that came into or out of your shielded screen room.

THEN 'HOW' DO WE USE THE MODEL 1 TRS-80 FOR RADIO FREQUENCY COMMUNICATIONS ? ? ?

Quite simply, Gridley. We take advantage of another of Mother Nature's immutable LAWS; i.e., the inverse square law that states: AS YOU MOVE AWAY FROM AN R.F. SOURCE THE POWER WILL DECREASE AS THE THE SQUARE OF THE DISTANCE; i.e., if the power level is 1 microwatt at one foot, at 2 feet it will be 1/(2)(2) = 1/4 microwatt. At 10 feet it will = 1/(10)(10) = 1/100th microwatt and at 50 feet distance it will = 1/(50)(50) = 1/2500th microwatt. Good old 'cut and try' testing, sometimes called the scientific empirical approach by those who wish to impress you with their expertise, has shown that once you get about 70 feet away from the TRS-80, the RFI amplitude

has dropped to the level where it will NOT interfere with MOST all devices including ham radio receivers. There are a few rules you MUST follow though if you wish to use this rather mundane yet very effective approach to solving the RFI problem. Let's take a look at the rules of this game.

1. The RFI generated by your TRS-80 must ONLY be able to reach your ham radio receiver via the external antenna you have mounted at LEAST 70 feet distant from the TRS-80. This means that NO RFI must be able to reach your receiver either by the common 120 volt ac power line OR by the direct route IF your ham radio receiver is poorly shielded OR by getting into the coax line between the external antenna and the receiver.

2. A good series of tests for these 3 VERY IMPORTANT and necessary requirements is to connect the receiver (most any cheapy USED ham receiver is ok) a few feet away from the TRS-80, plug it into the ac line AND connect your converter to the 70 feet of coax cable leading to the external antenna. DO NOT attach the antenna yet. Instead, plug in a temporary 50 ohm 1/4 or 1/2 watt resistor at the antenna end of the coax line and then turn your TRS-80 'on.' IF you cannot hear the RFI from your TRS-80 you have solved the problem; congratulations and skip the next paragraph. IF you can still hear the RFI, it has got to be getting into the converter/receiver via the power line, and/or direct radiation, and/or coax cable. Let's determine which one or all of them is doing the dirty work and solve the problem.

3. We will use the 'rose petal' or 'cabbage leaf' approach to the problem by removing one petal/leaf at a time. Solving one problem at a time is usually the ONLY way to successfully attack and do away with the RFI monster. First, remove the coax connector at the input to your VHF converter. IF the RFI completely disappears, THEN you are picking it up through the coax antenna lead. You now have two choices:

- Try rerouting the coax as far away from the TRS-80 as possible. IF this solves the problem, skip the rest of this paragraph.
- Purchase a better grade of RG8/U foam coax such as Belden #8214 FR-1 from Amateur Electronic Supply. A great deal of RG8/U coax has only MINIMAL copper shielding; i.e., it is not much better than zip cord.

IF you still are picking up RFI from the TRS-80, then it is either via direct radiation or via the ac power line. We suggest you eliminate the latter by installing an Electronic Specialists, Inc. model ISO-1A line isolator ($49.95) between the ac line feeding your TRS-80 and ham receiver. It contains 3 separate VERY effective filtered outlets that should isolate any and all RFI via this route. This should eliminate any ac line feed through of the RFI. IF you still have RFI you DO have a problem as it is most likely due to direct radiation from the TRS-80 getting into your receiver which MAY BE poorly shielded. About the only solution to this problem is to physically move the receiver AWAY from the TRS-80 till the level of stray RFI pickup is reduced to nil, OR to buy a better

grade used ham receiver with somewhat BETTER shielding. Remember, we will be feeding the output of our VHF converters to a cheapy USED ham receiver. More about this later.

ADDRESSES & PHONE NUMBERS FOR PARTS: (all accept VISA cards)

Amateur Electronic Supply 4828 West Fond du Lac Avenue Milwaukee, Wisconsin 53216	(RG8/U foam coax cable) (Cushcraft antennas) phone: 1-800-558-0411
Electronic Specialists, Inc. 171 South Main Street Natick, Massachusetts 01760	(ISO-1A line isolator) phone: (617)-655-1532
Hamtronics, Incorporated 65 Moul Road Hilton, New York 14468	(6 & 2 meter converters) phone: (716)-392-9430

BUILDING THE RECEIVING CONVERTERS FOR 6 AND 2 METER OPERATION:

For the purpose of illustration, we will use the ICOM 6 and 2 meter ALL MODE transceivers for the TRANSMIT segments of our data link and the Hamtronics (see address above) crystal controlled receiving converters at each end of the data link feeding its respective receiver. At the 6 meter TRANSMIT end of the data link we will be receiving on 2 meters and will use the Hamtronics model CA144(C) converter kit. At the 2 meter TRANSMIT end of the data link we will be receiving on 6 meters and will use the Hamtronics CA50(C) converter kit. Either kit sells for $39.95 with case and appropriate crystal and is a bargain considering the low price and high performance. Each kit can assembled by a novice in about 2 hours. The instructions are clear and concise. Should you have any problem they will correct it for you at modest charge. IF you do not wish to assemble either kit yourself, they are available at $54.95 wired and tested mounted in a case with connectors.

CHOOSING AN ANTENNA FOR 6 & 2 METERS:

May vary anywhere from the exotic to the extremely simple since we are only dealing with 'line of sight' transmission paths. Most any variety of antenna from two simple gound plane antennas at each end of the data link for each band on up to stacked 'BOOMER' multiple element LONG yagi arrays may used. It is up to you. For the 'build-it-yourself' type, there are many excellent 6 & 2 meter antennas covered in both the ARRL Antenna Book and ARRL Radio Amateur's VHF Manual. IF you are a newcomer to the amateur VHF bands and NOT inclined to build it yourself, we suggest you start off with reasonably small sized Cushcraft yagi antennas at each end for each band that can be mounted on most any variety of low cost TV antenna rotor. As such, you can use them for modest DXing (long distance work) as well as the computer data link too.

Both of the following Cushcraft 6 & 2 meter antennas are available from Amateur Electronic Supply, listed above.

Our recommendation for STARTERS for antennas would be:

6 Meters:

Cushcraft	A50-3	3 element	@ $ 54.95 or:
	A50-5	5 element	@ $ 74.95

2 Meters:

Cushcraft	A144-7	7 element	@ $ 32.95 or:
	A144-11	11 element	@ $ 44.95

All the above may be fed with any variety of 50 ohm coaxial transmission line, but we strongly recommend the Belden type mentioned on the previous page BOTH for its shield quality and low-loss foam insulation.

CONNECTING THE ICOM TRANSMITTER & HAMTRONICS CRYSTAL CONTROLLED CONVERTER AND RECEIVER TO YOUR MODEM:

Is quite simple if FIRST we beg-borrow-or-steal an old standard telephone handset. We will replace the microphone carbon element with a low cost crystal microphone element from Radio Shack #270-095 @ $1.99 and match it (sort of) to the 600 ohm input impedance of our ICOM transmitter by using a 1000 ohm to 8 ohm audio transformer, Radio Shack # 273-1380 @ $1.29 each. The transformer's 8 ohm speaker terminals will go to the ICOM mike connector and the 1000 ohm side to crystal mike element. The impedance match is certainly NOT exact, but good enough for our purposes. Use masking tape and cotton battens to hold the microphone element in place. In SOME telephone handsets there will be room to mount the transformer behind crystal mike. If NOT, simply tape the transformer to the back of the handset. Use mini-size shielded coax to run the wires from the microphone/matching transformer to the ICOM transmitter connector and the telephone receiver to your receiver's speaker terminals. The very LAST thing you want is stray 60 cycle ac hum or rf fields capacitively or inductively coupled into either line.

SELECTING A USED HAM RECEIVER FOR THE TUNEABLE I.F.:

Is probably the easiest part of the whole affair as there are zillions of cheapy ($100 price range) ham receivers floating around most everywhere. Models such as the 25 year old National NC-300 (we prefer) or Hallicrafters SX-71 abound at Hamfests for $100. and less, or may be located though the Ham-Ads in the back of QST Magazine published monthly by ARRL. All will tune the 29.000 MHz intermediate frequency we will be using as our 6 & 2 meter converters output frequency that represent 51 MHz and 145 MHz input, respectively.

It would be a wise idea IF you use an old used ham receiver as suggested above, to mount it AT least 8 to 10 feet AWAY from the TRS-80 due to the rather POOR shielding incorporated into these older models. DO NOT FORGET THE ISO-1A. It is a MUST.

LET'S SEE WHAT WE HAVE GOT HOOKED UP SO FAR:

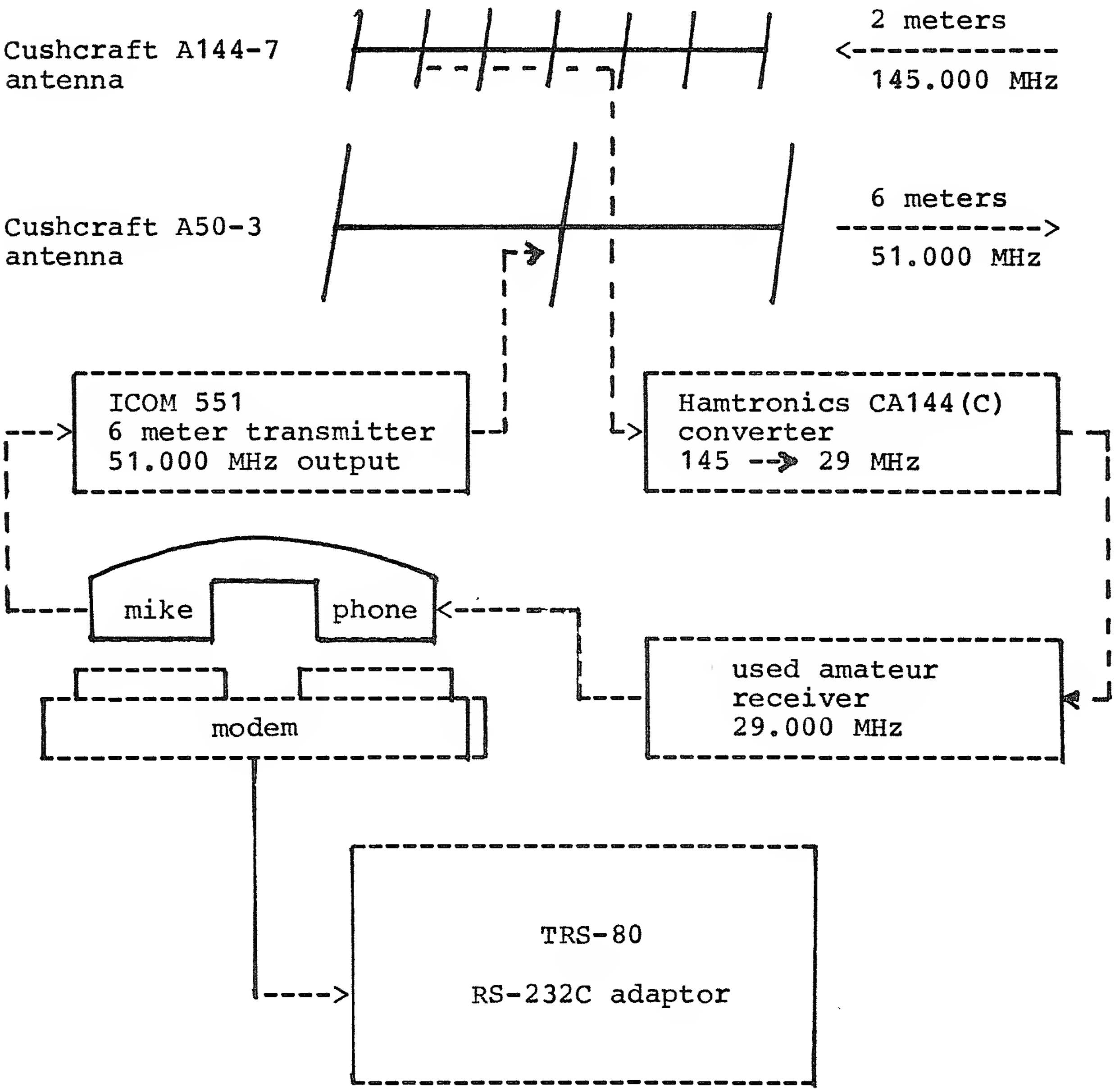


Figure 10-3

Figure 10-3 illustrates one end of the data link. The other end is quite similar and shown on the next page. There is ONE important facet of the hookup LEFT OUT of Figure 10-3; i.e., a low pass filter on the output of the ICOM 551 transmitter. Though the ICOM 551 includes a built-in low pass filter in its output line, it is recommended that you add an additional one just to be on the safe side and to prevent the 3rd harmonic of the 51 MHz transmitter = 153 MHz from de-sensitizing the Hamtronics 145 MHz converter. Either a homebrew filter listed in the ARRL Radio Amateur's VHF manual or a store bought variety such as the Barker & Williamson low power 6 meter type will work just fine. It is unnecessary to use a low pass filter on the system shown in Figure 10-4 as we are transmitting on 145.000 MHz and receiving on 51.000 MHz.

THE OTHER END OF OUR DATA LINK WOULD LOOK LIKE THIS:

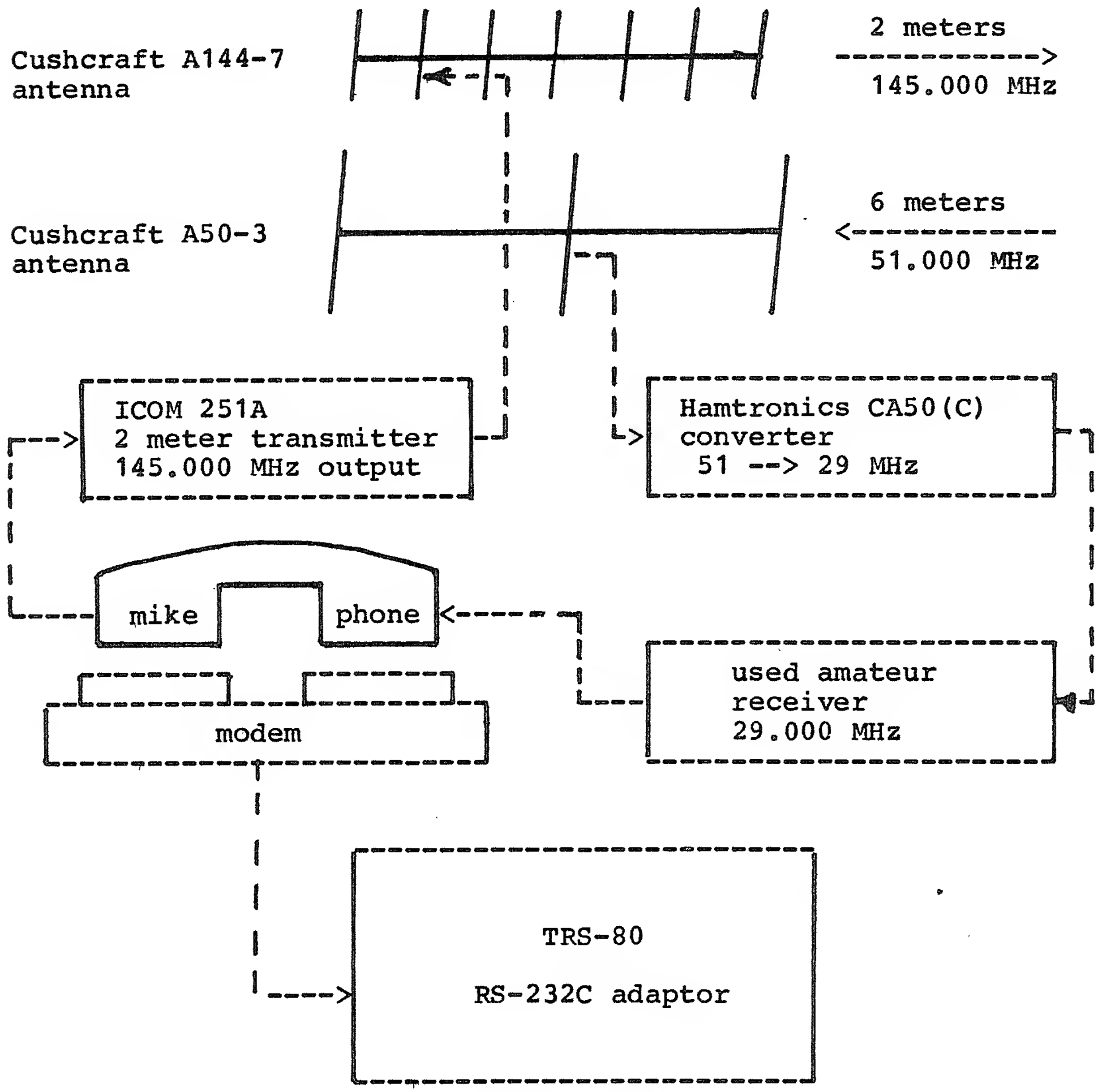


Figure 10-4

Here we have the other half of our 6 & 2 meter data link quite similar to that shown in Figure 10-3 EXCEPT the transmit and receive frequencies are reversed.

LET'S FIRE IT UP AND GET ON THE AIR ! ! !

Hold your horses, Gridley. Sure enough, we COULD fire it up and get on the air right now with most any variety of factory manufactured 1/2 duplex program, BUT you wouldn't have the slightest idea exactly WHAT was going on in your TRS-80. Do you want to be classified as a 'computer game freak' or do you really want to KNOW how this thing works, Gridley?

YOU MAY CALL ME ANYTHING, BUT A 'COMPUTER GAME FREAK.'

Very well, Gridley. We are going to write our own assembly language program and take one step at a time so you will indeed understand EXACTLY what is going on when we finally get ready to set up and operate the data link across Chautauqua Lake with our friend, Dr. Bill Laird at Dewittville Bay, New York. ANY complex appearing problem is really downtown simpleville when you disassemble it and approach it piece by piece. The complex appearing WHOLE is nothing more than the sum of many ridiculously simple parts.

THE RS-232C SERIAL INTERFACE ADAPTOR AND ITS PROTOCOLS:

I KNOW WHAT A PROTOTYPE AND PROCONSUL IS. WHAT'S A PROTOCOL?

Quite simply Gridley, it is 'the rules of etiquette' or somewhat simplified, 'the rules of the game' we must use when dealing with the TRS-80 RS-232C adaptor. This is the device we will be using to transform our TRS-80's parallel byte output to serial output for transmission over our half-duplex VHF radio data link. IF we understand EXACTLY how to program the RS-232C interface for the baud rate desired (110 or 300), the number of bits to be transmitted or received (7 or 8), number of stop bits to be used (1 or 2), and how to test for 'ready to transmit' and 'ready to receive' THEN writing an assembly language program to communicate between 2 TRS-80s via VHF data link will be downtown simplesville. Let's take it one step at a time.

THE RADIO SHACK TRS-80 RS-232C INTERFACE MANUAL #26-1145

Is a masterpiece of obfuscation, reportedly written in part by Dr. Rupert Kubala, the Ugandan Minister of Telecommunications.

SIR, PLEASE DON'T TALK DIRTY IN FRONT OF THE LADIES IN CLASS.

No Gridley, I am NOT talking dirty. To obfuscate means to obscure, confuse, or stupefy. The RS-232C manual mentioned above does just THAT to a fare-thee-well. A few cases in point would include: the circuit board shown on page 9 is upside down, the schematic shows the sense switches numbered backwards, and the entire manual appears to have been translated from Ugandan into English by a college student majoring in Egyptian hieroglyphics. What we are saying is that it reads somewhat less than clearly and straightforwardly, unless you enjoy cryptograms or crossword puzzles. We are told that Ed Juge-W5TOO had ABSOLUTELY nothing do with it and it was prepared in some haste to get it to market. Congratulations, Ed.

Before we get our feet wet, we had best mention that the RS-232C interface adaptor's connector to the expansion interface is lacking in reliability on ALL TRS-80s. This problem is easily cured by installing a REINFORCING bracket manufactured and sold by: Gunn Industries, 704 Franklin Blvd., Austin, Texas. This bracket is a MUST and sells for $5.00 ppd.

RS-232C INTERFACE BOARD SENSE SWITCHES:

Are completely UNNECESSARY and serve no useful purpose that we can imagine. Possibly Dr. Kubala was paid for both parts count and the length of his program on pages 23, 24, and 25 by the Hon. Idi Umin and Radio Shack. Henceforth, we will NOT use the configuration sense switches that may be read by INP(233). The important PORTS and their functions that we will be using in our VHF data link program with the RS-232C adaptor include:

PORT 232:
An OUT port 232 in either BASIC or assembly language serves to RESET the master data latch on the RS-232C interface board. The VALUE that we output to port 232 makes no nevermind. Any time we wish to change baud rate, number of bits transmitted or received, stop bits, parity even or odd, and parity on or off, we should first OUT 232,(any value) to put this latch in the condition necessary to accept the NEW data. An INP port 232 in either BASIC or assembly language tells us the condition of our modem's status register. We will NOT be using this function in this section, so will treat it with benign neglect.

PORT 233:
An OUT port 233,value in BASIC or assembly language sets the baud rate desired. We will be using either 110 baud or 300 baud so in BASIC or assembler, respectively, we will:

110 baud: OUT233, (BASIC) or OUT (233), (assembler)

300 baud: OUT233,85 (BASIC) or OUT (233),55H (assembler)

Of course, in assembly language we may use either decimal or hex, your choice depending on whether you have ten or sixteen fingers on your hands. An INP from this port also serves to tell us the cnfiguration of the RS-232C sense switches which we will ignore.

PORT 234:
An OUT port 234,value in either BASIC or assembler allows us too set: (MSB = most significant bit = bit 7, NMSB = next most significant bit, and LSB = least significant bit = bit zero).

BIT 7 (MSB)	:	1 for even parity and 0 for odd parity
BIT 6 (NMSB)	:	1 for 8 bits and 0 for 7 bits word length
BIT 5 (NMSB)	:	1 for 8 bits and '1' for 7 bits word length
BIT 4 (NMSB)	:	1 for 2 stop bits and 0 for 1 stop bit
BIT 3 (NMSB)	:	1 for parity OFF and 0 for parity ON
BIT 2 (NMSB)	:	1 enable transmit and 0 disable transmit
BIT 1 (NMSB)	:	1 request to send OFF & 0 request to send ON
BIT 0 (LSB)	:	1 terminal READY and 0 terminal NOT ready

We will NOT be using parity checking in the balance of this section, so BIT 7 is unimportant. Nevertheless, let's use EVEN = 1 so that we will not forget it when tying into communication bulletin board systems in another Chapter.

PORT 234 (continued):
An INP(234) and/or IN A,(234) tells us tthe conditon of UART status register as follows:

```
BIT 7 (MSB)  : 1 for data received & 0 for data NOT received
BIT 6 (NMSB) : 1 for transmit holding reg. empty & 0 for full
BIT 5 (NMSB) : 1 for an overrun error and 0 for NO error
BIT 4 (NMSB) : 1 for a framing error and 0 for NO error
BIT 3 (NMSB) : 1 for parity error and 0 for NO parity error.
BIT 2 (NMSB) : is not now used-saved for future applications.
BIT 1 (NMSB) : is not now used-saved for future applications.
BIT 0 (LSB)  : is not now used-saved for future applications.
```

We will be testing bits 7 and 6 later in this section. The rest we will ignore for the time being.

PORT 235:
Is the port with which we will be transmitting and receiving data. An OUT235,data or OUT (235),A will duly send the byte IF we first test the transmitter holding register for being ready by testing port 234's BIT 6 with an INP(234) then test the bit, or the IN A,(234) instruction and then test the bit. Receiving data is similar in that an INP(235) or IN A,(235) will accept the completed byte IF we first test BIT 7 of port 234 to ensure that WHOLE byte has been received. Everything takes TIME, and these tests for both the transmitted and received byte merely tell us whether or not the UART has completed its job of changing a parallel byte to a serial one with the appropriate protocol on transmit, and vice versa on receive.

A MODEST PROGRAM IN BASIC TO ILLUSTRATE A FEW POINTS:

BASIC is so very SLOW that we do not have to test EVERYTHING mentioned above to make it work. Set up your modem with the right hand switch in the TEST position and the left hand switch in the 'O' position. Most modems have SO MUCH audio output that they can hear themselves IF you fold a piece of paper lengthwise and lay it over the transmitter and receiver. If your modem is not that sensitive then place a phone in the modem's receptacles and dial any single number except nine or zero. IF on a party line, try it during the wee hours.

```
10 ' UART/MODEM DEMONSTRATION PROGRAM USING 110 OR 300 BAUD
15 ' 110 BAUD = 7 BITS - 2 STOP BITS - WITH PARITY DISABLED
20 ' 300 BAUD = 7 BITS - 1 STOP BIT  - WITH PARITY ENABLED
25 '
30 OUT232,255
35 INPUT"110 OR 300 BAUD DESIRED";X
40 IF X=110THENOUT233,34:OUT234,189:GOTO50
45 OUT233,85:OUT234,165
50 Y$=INKEY$:IFY$=""THEN50ELSEOUT235,ASC(Y$)
55 IF INP(234)<128GOTO55ELSE60
60 PRINTCHR$(INP(235));:GOTO50
```

Let's take it line by line. Note that we did not turn the expansion interface's clock OFF if you are using disk. The 25 millisecond 'sort of' interrupt is so short with respect to the 110 and 300 baud transmit and receive rate as to have little or no adverse effect. As such, the clock may be used to log ON or OFF as desired.

Line 30:
Resets our RS-232C master latch so that we may CHANGE the baud rate and other operating parameters.

Line 35:
Assigns either 110 or 300 to the variable X. Your choice.

Line 40:
IF you chose 110, then this line first sets the baud rate at 110 by outputting 34 (22H) to port 233. Port 234 is output the value 189 to set the following as 189 = 10111101 binary.

```
BIT 7 (MSB)  = 1 for even parity, though it won't be used
BIT 6 (NMSB) = 0 for 7 bit word length
BIT 5 (NMSB) = 1 for 7 bit word length
BIT 4 (NMSB) = 1 for 2 stop bits (most 110 baud use 2 stop)
BIT 3 (NMSB) = 1 for parity off (not needed for short paths).
BIT 2 (NMSB) = 1 to ENABLE transmit
BIT 1 (NMSB) = 0 request to send is ON
BIT 0 (LSB)  = 1 data terminal is READY
```

The program then goes to line 50 skipping over line 45.

Line 45:
IF you chose 300, then line 40 was skipped by the IF statement & the baud rate set at 300 by outputting 85 (55H) to port 233. Port 234 is output 165 = 10100101 binary to set as follows:

```
BIT 7 (MSB)  = 1 for even parity IF you wish to use it
BIT 6 (NMSB) = 0 for 7 bit word
BIT 5 (NMSB) = 1 for 7 bit word
BIT 4 (NMSB) = 0 for 1 stop bit (all 300 baud use 1 stop bit)
BIT 3 (NMSB) = 0 for parity ON since most CBBS use it
BITs 2, 1 & 0 same as above
```

Line 50:
Is the familar INKEY$ loop awaiting a character input from the keyboard which then outputs the ASCII value of the character to port 235 that is our UART's transmit port. It is unnecessary to check bit 6 of port 234 with INP for the transmit holding register EMPTY condition since the time it takes for BASIC to process lines 55's DATA RECEIVED status is much more than that required for everything to settle using simplex.

Line 55:
Does indeed check the DATA RECEIVED status by testing port 234. IF bit 7, the most significant bit DOES NOT = 1; i.e., less than 128 decimal = byte NOT received, then line 55 loops back again till it does. Once it does, then GOTO line 60.

Line 60:
Prints out the ASCII character. Then GOTOs the keyboard again.

I THOUGHT TODAY'S LECTURE WAS ON VHF DATA LINKS ? ? ?

Be patient, Gridley. We will get there in a moment. Looking back at our BASIC program on page 10-17, we MUST remember that we are only transmitting and receiving ONLY seven bits. What does that tell you, Gridley? Think about your answer a moment.

SEVEN IS MY LUCKY NUMBER. WHY NOT ? ? ?

Well Gridley, how would you transmit or receive a BASIC or assembly language program IF the highest number you could use was 7 bits long? After all, in hex that only = 127?

NOT VERY EASILY ! ! ! I DIDN'T THINK ABOUT THAT ! ! !

A clear and lucid statement, Gridley. Thank you. Let's try writing an assembly language program that will do all these good things for us:

- Allow us to select baud rates of either 110 or 300, as we did in the previous program.
- Set byte length at 8 bits and IF 110 baud THEN 2 stop bits or IF 300 baud THEN 1 stop bit.
- Transmit and receive half-duplex with input from either keyboard displaying its output on video, thus NOT requiring the echo function of full-duplex to see what was tranmitted.
- Ability to transmit and receive either BASIC, source, or object code programs from one terminal to another.
- When transmitting programs from one station to another, a transmit AND receive byte counter at each end of the data link to allow relatively simply cross checking without the time consuming full-duplex echo function.
- Receiving station operator selection of WHERE in memory to store the received program, thus allowing most any MEM size receiving station to work another of a different size.
- Video display at both the transmitting and receiving TRS-80 of the decimal value of each byte transmitted or received, respectively.
- Ability to switch to BASIC from the half-duplex communication program with single or multiple keyboard entry.
- Future growth capability to use full-duplex if desired AND check not just the parity of the byte transmitted and reeived, BUT the absolute decimal value of EACH byte and re-transmit that byte until the correct byte is received and correctly acknowledged.

Now let's take a look at this program's source code on pages 10-20 to 10-22 and object code on pages 10-23 to 10-25.

```
00100 ; W4UCH TRS-80 VHF DATA LINK PROGRAM   -   VHF 1 & 2
00110 ;
00120 ; WILL TRANSMIT AND RECEIVE KEYBOARD DATA VIA DUPLEX
00130 ;
00140 ; PLUS TRANSFER BASIC AND ASSEMBLER PROGRAM LISTINGS
00150 ;
00160 W4UCH   EQU     32000           ;= 7D00H FOR PURISTS ?
00170         ORG     W4UCH           ;START THE PROGRAM HERE
00180         CALL    01C9H           ;CLS ROM SUBROUTINE
00190         LD      SP,32760        ;MOVE STACK HERE
00200         OUT     (232),A         ;UART MASTER RESET LATCH
00210         LD      HL,MESS1        ;STRING MEM ADDRESS
00220         CALL    28A7H           ;DISPLAY STRING ROUTINE
00230 MESS1   DEFM    'A = ONE HUNDRED TEN BAUD & B = THREE HUNDRED BAU
00240         DEFB    00              ;END OF MESSAGE DELIMITER
00250         CALL    049H            ;KEYBOARD INPUT TO 'A'
00260         CALL    032AH           ;DISPLAY INPUT
00270         CP      65              ;A=65 SUBTRACT FROM 'A'
00280         JR      Z,LOAD          ;GOTO LOAD IF ZERO
00290         LD      A,13            ;13 = CONTROL SKIP A LINE
00300         CALL    033H            ;DO IT ON VIDEO DISPLAY
00310         LD      A,55H           ;55H = 300 BAUD
00320         OUT     (233),A         ;SET BAUD RATE AT 300
00330         LD      A,237           ;8 BITS, 1 STOP, NO PAR
00340         OUT     (234),A         ;INITIALIZE ABOVE
00350 LOOK    LD      A,14            ;CONTROL 14 = CURSOR 'ON'
00360         CALL    033H            ;DO IT ON VIDEO
00370         LD      A,(14400)       ;= KEYBOARD CLEAR KEY ROW
00380         CP      2               ;2 = CLEAR KEY PRESSED
00390         JP      Z,W4UCH         ;CHANGE BAUD RATE
00400         CP      4               ;4 = 'BREAK' KEY PRESSED
00410         JP      Z,XMIT2         ;GOTO XMIT2 SUBROUTINE
00420         CP      6               ;'CLEAR & BREAK' PRESSED?
00430         JP      Z,0072H         ;RETURN TO BASIC 'READY'
00440         CALL    INPUT           ;INPUT SUBROUTINE
00450         CALL    OUTPUT          ;OUTPUT SUBROUTINE
00460         JP      LOOK            ;TAKE ANOTHER LOOK
00470 INPUT   IN      A,(234)         ;IS UART READY ?
00480         BIT     7,A             ;TEST BIT 7 SET 'Z'
00490         RET     Z               ;RETURN 'IF' NOT READY
00500         IN      A,(235)         ;LOAD PORT 235 INTO 'A'
00510         JP      Z,INPUT         ;IF ZERO, LOOK AGAIN
00520         CALL    033H            ;IF NOT ZERO, DISPLAY IT
00530         JP      INPUT           ;LOOK AGAIN
00540 OUTPUT  CALL    02BH            ;CHECK THE KEYBOARD
00550         OR      A               ;SET 'Z' FLAG
00560         RET     Z               ;RETURN IF ZERO
00570         PUSH    AF              ;SAVE AF IN STACK
00580 XMIT1   IN      A,(234)         ;TEST UART 'READY'
00590         BIT     6,A             ;TEST BIT 6 SET 'Z'
00600         JP      Z,XMIT1         ;IF 'NOT' LOOK AGAIN
```

```
00610          POP     AF              ;RESTORE FROM STACK
00620          OUT     (235),A         ;SEND BYTE TO UART
00630          RET                     ;RETURN TO 'LOOP'
00640 LOAD     LD      A,13            ;13 = CONTROL SKIP A LINE
00650          CALL    033H            ;DO IT ON VIDEO
00660          LD      A,22H           ;22H = 110 BAUD
00670          OUT     (233),A         ;SET BAUD RATE TO 110
00680          LD      A,253           ;8 BITS, 2 STOP, NO PAR
00690          OUT     (234),A         ;INITIALIZE ABOVE
00700          JP      LOOK            ;GOTO LOOK AGAIN
00710 ;
00720 ; BASIC & ASSEMBLER PGM TRANSFER VIA VHF DATA LINK
00730 ;
00740 XMIT2    LD      A,13            ;CONTROL 13 = SKIP A LINE
00750          CALL    033H            ;DO IT ON VIDEO
00760          LD      HL,MESS2        ;MESSAGE 2 ADDRESS TO HL
00770          CALL    28A7H           ;DISPLAY STRING ROUTINE
00780 MESS2    DEFM    'INPUT PROGRAM BEGINNING ADDRESS  ?  '
00790          DEFB    0               ;END OF MESSAGE DELIMITER
00800          CALL    1BB3H           ;KYBD/VIDEO INPUT ROUTINE
00810          RST     10H             ;SCAN STRING - SET C FLAG
00820          CALL    0E6CH           ;ASCII - ACCUM RET MIN
00830          CALL    0A7FH           ;CONVERT ACCUM TO INTEGER
00840          PUSH    HL              ;SAVE ACCUM IN STACK
00850          LD      HL,MESS3        ;MESSAGE 3 ADDRESS TO HL
00860          CALL    28A7H           ;DISPLAY STRING ROUTINE
00870 MESS3    DEFM    'INPUT NO. OF PROGRAM BYTES TO MOVE ?  '
00880          DEFB    0               ;END OF MESSAGE DELIMITER
00890          CALL    1BB3H           ;KYBD/VIDEO INPUT ROUTINE
00900          RST     10H             ;SCAN STRING - SET C FLAG
00910          CALL    0E6CH           ;ASCII - ACCUM RET MIN
00920          CALL    0A7FH           ;CONVERT ACCUM TO INTEGER
00930          PUSH    HL              ;SAVE NO. TO MVE IN STACK
00940          LD      HL,MESS4        ;MESSAGE 4 ADDRESS TO HL
00950          CALL    28A7H           ;DISPLAY STRING ROUTINE
00960 MESS4    DEFM    'TRANSMIT OR RECEIVE (T/R)  ?  '
00970          DEFB    0               ;END OF MESSAGE DELIMITER
00980          CALL    049H            ;AWAIT KEYBOARD INPUT
00990          CP      82              ;R=82 SUBTRACT FROM 'A'
01000          JP      Z,TELL1         ;GOTO TELL IF 'R' PRESSED
01010          POP     DE              ;RESTORE NO. BYTES TO MVE
01020          POP     HL              ;RESTORE BEGIN ADDRESS
01030 PORT     IN      A,(234)         ;IS UART READY ?
01040          BIT     6,A             ;TEST BIT 6 SET 'Z' FLAG
01050          JP      Z,PORT          ;GO BACK FOR ANOTHER LOOK
01060          LD      A,(HL)          ;HL MEM LOCATION TO 'A'
01070          OUT     (235),A         ;OUTPUT VIA UART AT 235
01080          INC     HL              ;+1 TO MEMORY LOCATION
01090          DEC     DE              ;-1 TO BYTE COUNTER
01100          PUSH    DE              ;SAVE IN THE STACK
01110          PUSH    HL              ; "    "    "    "
01120          CALL    SHOW            ;GOTO SHOW SUBROUTINE
```

```
01130        POP   HL          ;RESTORE HL FROM STACK
01140        POP   DE          ; "      DE   "    "
01150        LD    A,D         ;PART OF ZERO TEST
01160        OR    E           ;OR 'A' WITH 'E'
01170        JP    NZ,PORT     ;GOTO PORT TILL ZERO
01180 WAIT   LD    A,(14400)   ;KEYBOARD 'SPACE' BAR ROW
01190        CP    128         ;SPACE BAR 'PRESSED' ?
01200        JP    NZ,WAIT     ;GOTO WAIT TILL PRESSED
01210        JP    LOOK        ;RETURN TO KYBD PROGRAM
01220 SHOW   LD    L,A         ;LD L REG WITH BYTE VALUE
01230        LD    A,0         ;LD A REGISTER WITH ZERO
01240        LD    H,A         ;ZERO OUT H REGISTER
01250        CALL  0A9AH       ;MOVE HL INTO ACCUM
01260        CALL  0FBDH       ;CONVERT ACCUM TO STRING
01270        CALL  28A7H       ;DISPLAY STRING ON VIDEO
01280        LD    A,(14400)   ;KEYBOARD 'BREAK' ROW
01290        CP    4           ;IS 'BREAK' PRESSED ?
01300        JP    Z,LOOK      ;RETURN TO KYBD PROGRAM
01310        RET               ;RET FOR NEXT CHARACTER
01320 TELL1  POP   DE          ;RESTORE NO. BYTES TO MVE
01330        POP   HL          ;RESTORE BEGIN ADDRESS
01340        DEC   HL          ;ADJUST RECV MEM LOCATION
01350 TELL2  IN    A,(234)     ;UART STATUS REGISTER
01360        BIT   7,A         ;TEST BIT 7 SET Z FLAG
01370        JP    Z,TELL2     ;LOOK AGAIN 'NOT' READY
01380        IN    A,(235)     ;INCOMING BYTE PORT
01390        JP    Z,TELL2     ;LOOK AGAIN IF 'ZERO'
01400        LD    (HL),A      ;STASH BYTE IN MEMORY
01410        INC   HL          ;+1 TO MEMORY LOCATION
01420        DEC   DE          ;-1 TO PGM BYTE COUNTER
01430        PUSH  DE          ;SAVE IN THE STACK
01440        PUSH  HL          ; "    "   "    "
01450        CALL  SHOW        ;GOTO SHOW SUBROUTINE
01460        POP   HL          ;RESTORE HL FROM STACK
01470        POP   DE          ; "      DE   "    "
01480        LD    A,D         ;PART OF ZERO TEST
01490        OR    E           ;OR 'A' WITH 'E' REGS
01500        JP    NZ,TELL2    ;GOTO TELL IF NOT ZERO
01510 HOLD   LD    A,(14400)   ;KEYBOARD 'SPACE' BAR ROW
01520        CP    128         ;SPACE BAR 'PRESSED' ?
01530        JP    NZ,HOLD     ;GOTO HOLD TILL PRESSED
01540        JP    LOOK        ;GOTO KEYBOARD PROGRAM
01550        END   W4UCH       ;EL FIN = EL BEGUINE
```

- Source Code -

```
7D00               00160 W4UCH   EQU     32000
7D00               00170         ORG     W4UCH
7D00 CDC901        00180         CALL    01C9H
7D03 31F87F        00190         LD      SP,32760
7D06 D3E8          00200         OUT     (232),A
7D08 210E7D        00210         LD      HL,MESS1
7D0B CDA728        00220         CALL    28A7H
7D0E 41            00230 MESS1   DEFM    'A = ONE HUNDRED TEN BAUD &
B = THREE HUNDRED BAUD ? '
7D42 00            00240         DEFB    00
7D43 CD4900        00250         CALL    049H
7D46 CD2A03        00260         CALL    032AH
7D49 FE41          00270         CP      65
7D4B 284E          00280         JR      Z,LOAD
7D4D 3E0D          00290         LD      A,13
7D4F CD3300        00300         CALL    033H
7D52 3E55          00310         LD      A,55H
7D54 D3E9          00320         OUT     (233),A
7D56 3EED          00330         LD      A,237
7D58 D3EA          00340         OUT     (234),A
7D5A 3E0E          00350 LOOK    LD      A,14
7D5C CD3300        00360         CALL    033H
7D5F 3A4038        00370         LD      A,(14400)
7D62 FE02          00380         CP      2
7D64 CA007D        00390         JP      Z,W4UCH
7D67 FE04          00400         CP      4
7D69 CAAB7D        00410         JP      Z,XMIT2
7D6C FE06          00420         CP      6
7D6E CA7200        00430         JP      Z,0072H
7D71 CD7A7D        00440         CALL    INPUT
7D74 CD8A7D        00450         CALL    OUTPUT
7D77 C35A7D        00460         JP      LOOK
7D7A DBEA          00470 INPUT   IN      A,(234)
7D7C CB7F          00480         BIT     7,A
7D7E C8            00490         RET     Z
7D7F DBEB          00500         IN      A,(235)
7D81 CA7A7D        00510         JP      Z,INPUT
7D84 CD3300        00520         CALL    033H
7D87 C37A7D        00530         JP      INPUT
7D8A CD2B00        00540 OUTPUT  CALL    02BH
7D8D B7            00550         OR      A
7D8E C8            00560         RET     Z
7D8F F5            00570         PUSH    AF
7D90 DBEA          00580 XMIT1   IN      A,(234)
7D92 CB77          00590         BIT     6,A
7D94 CA907D        00600         JP      Z,XMIT1
7D97 F1            00610         POP     AF
7D98 D3EB          00620         OUT     (235),A
7D9A C9            00630         RET
7D9B 3E0D          00640 LOAD    LD      A,13
7D9D CD3300        00650         CALL    033H
7DA0 3E33          00660         LD      A,22H
7DA2 D3E9          00670         OUT     (233),A
```

```
7DA4 3EFD     00680          LD      A,253
7DA6 D3EA     00690          OUT     (234),A
7DA8 C35A7D   00700          JP      LOOK
7DAB 3E0D     00740 XMIT2    LD      A,13
7DAD CD3300   00750          CALL    033H
7DB0 21B67D   00760          LD      HL,MESS2
7DB3 CDA728   00770          CALL    28A7H
7DB6 49       00780 MESS2    DEFM    'INPUT PROGRAM BEGINNING ADD
RESS  ?  '
7DDA 00       00790          DEFB    0
7DDB CDB31B   00800          CALL    1BB3H
7DDE D7       00810          RST     10H
7DDF CD6C0E   00820          CALL    0E6CH
7DE2 CD7F0A   00830          CALL    0A7FH
7DE5 E5       00840          PUSH    HL
7DE6 21EC7D   00850          LD      HL,MESS3
7DE9 CDA728   00860          CALL    28A7H
7DEC 49       00870 MESS3    DEFM    'INPUT NO. OF PROGRAM BYTES
TO MOVE ?  '
7E12 00       00880          DEFB    0
7E13 CDB31B   00890          CALL    1BB3H
7E16 D7       00900          RST     10H
7E17 CD6C0E   00910          CALL    0E6CH
7E1A CD7F0A   00920          CALL    0A7FH
7E1D E5       00930          PUSH    HL
7E1E 21247E   00940          LD      HL,MESS4
7E21 CDA728   00950          CALL    28A7H
7E24 54       00960 MESS4    DEFM    'TRANSMIT OR RECEIVE (T/R)
?  '
7E42 00       00970          DEFB    0
7E43 CD4900   00980          CALL    049H
7E46 FE52     00990          CP      82
7E48 CA867E   01000          JP      Z,TELL1
7E4B D1       01010          POP     DE
7E4C E1       01020          POP     HL
7E4D DBEA     01030 PORT     IN      A,(234)
7E4F CB77     01040          BIT     6,A
7E51 CA4D7E   01050          JP      Z,PORT
7E54 7E       01060          LD      A,(HL)
7E55 D3EB     01070          OUT     (235),A
7E57 23       01080          INC     HL
7E58 1B       01090          DEC     DE
7E59 D5       01100          PUSH    DE
7E5A E5       01110          PUSH    HL
7E5B CD707E   01120          CALL    SHOW
7E5E E1       01130          POP     HL
7E5F D1       01140          POP     DE
7E60 7A       01150          LD      A,D
7E61 B3       01160          OR      E
7E62 C24D7E   01170          JP      NZ,PORT
7E65 3A4038   01180 WAIT     LD      A,(14400)
7E68 FE80     01190          CP      128
7E6A C2657E   01200          JP      NZ,WAIT
```

```
7E6D C35A7D    01210          JP      LOOK
7E70 6F        01220 SHOW     LD      L,A
7E71 3E00      01230          LD      A,0
7E73 67        01240          LD      H,A
7E74 CD9A0A    01250          CALL    0A9AH
7E77 CDBD0F    01260          CALL    0FBDH
7E7A CDA728    01270          CALL    28A7H
7E7D 3A4038    01280          LD      A,(14400)
7E80 FE04      01290          CP      4
7E82 CA5A7D    01300          JP      Z,LOOK
7E85 C9        01310          RET
7E86 D1        01320 TELL1    POP     DE
7E87 E1        01330          POP     HL
7E88 2B        01340          DEC     HL
7E89 DBEA      01350 TELL2    IN      A,(234)
7E8B CB7F      01360          BIT     7,A
7E8D CA897E    01370          JP      Z,TELL2
7E90 DBEB      01380          IN      A,(235)
7E92 CA897E    01390          JP      Z,TELL2
7E95 77        01400          LD      (HL),A
7E96 23        01410          INC     HL
7E97 1B        01420          DEC     DE
7E98 D5        01430          PUSH    DE
7E99 E5        01440          PUSH    HL
7E9A CD707E    01450          CALL    SHOW
7E9D E1        01460          POP     HL
7E9E D1        01470          POP     DE
7E9F 7A        01480          LD      A,D
7EA0 B3        01490          OR      E
7EA1 C2897E    01500          JP      NZ,TELL2
7EA4 3A4038    01510 HOLD     LD      A,(14400)
7EA7 FE80      01520          CP      128
7EA9 C2A47E    01530          JP      NZ,HOLD
7EAC C35A7D    01540          JP      LOOK
7D00           01550          END     W4UCH
00000 TOTAL ERRORS

HOLD    7EA4 01510   01530
INPUT   7D7A 00470   00440 00510 00530
LOAD    7D9B 00640   00280
LOOK    7D5A 00350   00460 00700 01210 01300 01540
MESS1   7D0E 00230   00210
MESS2   7DB6 00780   00760
MESS3   7DEC 00870   00850
MESS4   7E24 00960   00940
OUTPUT  7D8A 00540   00450
PORT    7E4D 01030   01050 01170
SHOW    7E70 01220   01120 01450
TELL1   7E86 01320   01000
TELL2   7E89 01350   01370 01390 01500
W4UCH   7D00 00160   00170 00390 01550
WAIT    7E65 01180   01200
XMIT1   7D90 00580   00600
XMIT2   7DAB 00740   00410
```

- Object Code -

THAT SURE SEEMS LIKE A LOT OF PROGRAM TO ME ! ! !

Actually it is only 430 bytes long, Gridley. Even those TRS-80 users with only 16K MEM, disk, and NEWDOS+ or NEWDOS 80 will still have considerable room between 26810 MEM and 32000 MEM = 5190 bytes available for other programming, though we suggest that they purchase the extra 16K or 32K MEM for the expansion interface. A new LOW in the price of RAM memory was scored during fall 1980 with 16K of 8 bit RAM chips advertised for under $50. by a number of suppliers. Just be CAREFUL and make sure that the bargain chips are not SECONDS that are TOO SLOW to work with your TRS-80. They should be GUARANTEED.

We have tried to make the comments in this program self-explanatory, for the most part. As in previous programs, we have intermingled hex with decimal which tends to drive self proclaimed purists up the wall. So be it. Feel free to use whichever number system is most convenient for YOU. Most CALLs to ROM are more easily remembered in hex as most addresses in RAM are more easily remembered in decimal. It really makes no nevermind AS LONG AS THE PROGRAM WORKS. The latter item is the TRUE test of a well written program as long as the comments are in English for English speaking readers, French for French speaking readers, and German for German speaking readers. Volumes 1, 2, and 3 are now available in French and German language editions for those so inclined.

Let's expand upon a few of the program's comments before hooking it up to our amateur radio transmitter and receiver and going on the air.

Line 200:
Resets our RS-232C board's UART latch so we may set it up for a new baud rate, plus even or odd parity, number of bits (word length), number of stop bits, and parity ON or OFF, etc.

Line 280:
Sends the program off to LOAD in line 640 IF we selected 110 baud by inputting 'B'. Lines 650 and 660 set the 110 baud rate, with lines 670 and 680 setting even parity (though we do not use it), the number of bits (word length) are set at 8, TWO stop bits, and parity OFF, before GOTO LOOK in line 350.

Lines 310-340:
Do much the same as above, except now the baud rate is set at 300 IF you selected 'B'. Parity is set even (though not used), the number of bits (word length) are set at 8, ONE stop bit, and parity OFF, before falling through to LOOK.

Lines 350-360:
Provide a 'blinking cursor' as the program cycles through the LOOK, and INPUT & OUTPUT program sequences. IF you do not favor a 'blinking cursor,' by all means remove these two lines or change the cursor to any graphics character desired. We like it as it is the same blinking cursor we have used the last two years with our modified disk Electric Pencil (tm).

Lines 370-460:
Are checked every 'sweep' through LOOK to allow us to change baud rate = CLEAR key pressed, GOTO the transmit or receive a program subroutine = BREAK key pressed, or return to BASIC with a 'READY' = both CLEAR and BREAK keys pressed. Since our RS-232C adaptor does not use either of these keys for any purpose, NO functions are impaired or modified.

Line 470-630:
Are the real work horses of our transmit and receive program when the data is being sent from either station's keyboard. The comments are largely self-explanatory and straightforward with NO sneaky tricks involved.

Lines 740-1020:
Deserve a brief explanation. Before we can transmit a program (or ANY data from memory) we need to know its beginning address which is PUSHed into the stack by line 840. If we are on the receiving end of the transmission the program wants to know WHERE you wish the program to be loaded in MEM. IF it is a BASIC program, we would use 17129 without disk and 26810 with NEWDOS+ or NEWDOS 80 on disk. IF it is an object code program, we know exactly where we put it. IF it is a source code program loaded into MEM from disk it will probably begin between 29000 and 30300 in MEM. Use this mini-BASIC program to find the beginning and end AFTER YOU LOAD IT INTO MEM:

```
10 FORX=29000TO35000:IFX>32767THENX=32767-65535
20 Y=PEEK(X):IFY>127GOTO50
30 IFY<32GOTO50
40 PRINTCHR$(Y);:GOTO60
50 PRINTY;
60 NEXT
```

Assuming you use the PEEK function to ascertain the END of a BASIC program (do not forget to include the double zeros at the end) and you know the last byte MEM location of an object code program, it is easy to answer Line 870's question, 'INPUT NO. OF BYTES TO MOVE.' You should enter the SAME number at both the transmit and receive ends of the data link.

The next question from line 960, 'TRANSMIT OR RECEIVE (T/R)' is an easy one to answer. NOTE: we used PUSH HL in both lines 840 and 930 stash the data into the stack, BUT used POP DE and POP HL in line 1010 and 1020 to POP the stack into the proper registers.

Lines 1030-1310:
Are similar to our OUTPUT lines 540-630 except that now the program also increments HL which is our MEM location counter and decrements the bytes transmitted in lines 1080 and 1090 before CALLing SHOW that is in line 1220.

Lines 1220-1270:
Are a rather simple way to display on video the decimal value of the byte either transmitted or received using the ACCUM.

We use the ACCUM, string conversion, and string display CALLs to output the transmitted OR received byte on video at BOTH ends of our data link. Since we are NOT echoing the received byte as in a full-duplex program or checking for parity this may serve as a simple visual check (if desired) for the validity of the received program. Once you gain confidence in the program's and your communications equipment's & path's reliability, you may eliminate these lines.

Lines 1280-1310:
Allow you to 'escape' from either the transmit or receive subroutines by pressing the BREAK key at anytime. This is a handy function when you have mistakenly entered the wrong starting address or program length earlier.

Lines 1320-1500:
Do the same thing at the receiving end of the data link as lines 1030-1170 accomplished for the transmit end. First, the number of bytes to move and beginning address are POPped off the stack into registers DE & HL. Line 1340 is a sneaky way of correcting the starting address by minus one since we were too lazy unload the holding register earlier. Lines 1350 to 1390 setup the received character in 'A' register when all 8 bits have been duly received and converted from serial to parallel by the UART. Line 1400 stashes the received byte in the appropriate MEM location and lines 1410-1420 increment the MEM location +1 and decrement the byte counter by -1. Gridley has his hand up. Question, Gridley?

WHY DIDN'T YOU USE THE LDIR INSTRUCTION ? ? ?

Another good question. We probably could have Gridley, but the program is a bit easier to follow in this format for those not as advanced as you.

The incremented MEM location and decremented byte counter are then saved in the stack by lines 1430-1440 before CALLing SHOW to display the byte on video. Upon RETurning from SHOW, lines 1460 to 1500 test the byte counter for zero and IF all bytes have been received then falls through to line 1510.

Lines 1510-1540:
Allow us to observe the last page (if a long program) on video after the whole program has been received. It waits for us to press the SPACE bar before returning to normal keyboard control. You will note that it is a duplicate of the WAIT sequence in lines 1180-1210, so you may save a few bytes by changing line 1510 to: JP WAIT, and eliminating lines 1520-1540.

WHY DO YOU USE JPs INSTEAD OF JRs FOR SHORT JUMPS ? ? ?

A logical question, Gridley. It is a trade-off between MEM required and TIME. Sure, JRs save 1 byte of MEM compared to JPs, but they take 20% MORE time for the Z-80 to process. It is really up to you; MEM or TIME, your choice. MEM is cheap.

RUNNING THE HALF-DUPLEX PROGRAM ON THE 6 & 2 METER BANDS:

The following is a typical exchange between the author's ham station-W4UCH/2 and Dr. Bill Laird's ham station-W2CIX, across Lake Chautauqua. W4UCH will be transmitting on 51.000 MHz and receiving on 145.000 Mhz while W2CIX will be tranmitting on 145.000 MHz and receiving on 51.000 MHz, thus allowing half-duplex operation. We will set our mechanical timer at 10 minutes to remind us to identify our stations using voice or CW to meet F.C.C. requirements. All we have to do on voice is take our phones out of the cradles and say, "W2CIX this W4UCH/2 for ident," and then Bill does the same. It is a good idea to do it simultaneously as voice into the MODEM produces some rather weird garbage from the program IF the other phone is ON the MODEM. Another question, Gridley?

YES SIR. I HAVE A SNEAKY FEELING THIS PROGRAM WOULD WORK ON AN ORDINARY TELEPHONE LINE AS WELL AS ON THE AMATEUR BANDS ! !

Right again, Gridley. Our MODEMs could care less WHERE their output goes and WHERE the input comes from. But, since this Chapter is about RADIO teletype (tm) let's stick with RTTY for the time being. IF you want to work a friend's TRS-80 across town on the phone line, by all means do so. This program will work just as well. Our particular location is such that to work Dr. Laird across the lake is a long distance call. Why bother with toll charges since we are both radio amateurs and TRS-80 buffs?

About the only precautions we need to take are to adjust the volume on the transmitters' mike inputs and receivers' audio outputs to the level where they do NOT overload their respective inputs. This is a fairly simple cut and try bit that may be done in a few moments. NOTE: the microphone and transformer mentioned earlier for installation in the telephone handset was replaced with a 600 ohm dynamic microphone element that happened to fit into the handset. It was from an unidentified mike of Japanese origin. Two were purchased for 50 cents each at a local hamfest.

FOLLOWING IS OUR FIRST EXCHANGE ON THE 6 AND 2 METER BANDS AS SEEN FROM THE W4UCH/2 END OF THE DATA LINK:

```
A = ONE HUNDRED TEN BAUD & B = THREE HUNDRED BAUD ? B

W2CIX THIS IS W4UCH/2. DO YOU COPY BILL?

SURE DO. GO AHEAD.

AH, THE MIRACLES OF MODERN SCIENCE. SONOFAGUN, I JUST MISTYPED I
N A LETTER AND THE BACKSPACE WORKS GREAT. OVER

OK ROBERTO. IF YOU HAD DONE THAT ON AN ORDINARY RADIO TELETYPE I
T WOULD HAVE JUST OVERPRINTED AND LEFT A REAL MESS. W4UCH/2 DE W
2CIX
```

FB WILLIAM. SINCE WE ARE USING ALL 64 SPACES ON THE VIDEO DISPL
AY THIS WILL PRINTOUT 2 SPACES WIDER ON THE MANUSCRIPT. I AM USI
NG THE 'JKL' NEWDOS + FUNCTION TO SAVE THESE TRANSMISSIONS FOR P
OSTERITY. LET ME TRY SENDING YOU THE FIRST FIFTY BYTES OF THIS
PROGRAM. ALL YOU NEED DO IS TO PRESS THE 'BREAK' KEY TO GET INTO
THE TRANSFER DATA MODE. THEN ENTER 30000 FOR THE BEGINNING ADDRE
SS ADDRE SS, 50 FOR BYTES TO MOVE, AND PRESS THE 'R' KEY. OVER

OK BOB. WILL DO. THEN I'LL JUST WAIT FOR YOU TO SEND. K

INPUT PROGRAM BEGINNING ADDRESS ? ? 32000
INPUT NO. OF PROGRAM BYTES TO MOVE ? ? 50
TRANSMIT OR RECEIVE (T/R) ? 205 201 1 49 248 127 211 232 33 1
4 125 205 167 40 65 32 61 32 79 78 69 32 72 85 78 68 82 69 68 32
84 69 78 32 66 65 85 68 32 38 32 66 32 61 32 84 72 82 69 69

NOTE: ABOVE IS THE WAY THE TRANSMIT END WAS SET UP AND THE BYTES
DISPLAYED ON THE TRANSMIT VIDEO.

AT THE RECEIVING END - IGNORE THE FIRST BYTE WHICH IS PLACED IN
THE DESIRED MEM LOCATION 'MINUS' ONE. NOW LET'S RETURN TO THE MA
NUSCRIPT AFTER THE FUN AND GAMES. MANY THANKS WILLIAM. W2CIX DE
W4UCH/2 OFF AND CLEARING THE FREQUENCY. K

At the other end of our VHF data link, the data came across exactly as shown on the preceeding page except for the FIRST decimal number after the "TRANSMIT OR RECEIVE (T/R)" shown on the video display. This number was ignored by the program which stored the CORRECT data beginning at MEM location 30000. That is why the DECrement was included in line 1420.

Had this been a BASIC program we would have started at 17129 MEM location for non-disk and BASIC2 or at 26810 MEM location AT BOTH ENDS OF THE DATA LINK if we were using disk with NEWDOS+ or NEWDOS 80. Once the BASIC program has been transferred from one station to the other it may RUN just like any other BASIC program and/or stored on cassette or disk just like any other BASIC program. IF you are loading it to cassette do NOT forget to turn your clock OFF with 'CMD"R"' first.

CONCLUSION OF SECTION ONE - CHAPTER 10:

I SHOULD HOPE SO. THIS HAS BEEN THE LONGEST SECTION ON RECORD!

Come on, Gridley. You stayed on the air last night till after midnight with W2CIX exchanging programs with each other. I am sure you two would still be at it if your Mother had not finally pulled the fuses on your ham shack and TRS-80.

What we have tried to illustrate is how very EASILY one may use their TRS-80 to communicate with another even IF you choose to write your own program and choose to use the amateur radio VHF bands for duplex operation. Remember, any supposedly complex system is NOTHING more than a bunch of SIMPLE parts.

SECTION II - CHAPTER 10

THE MACROTRONICS M-80 HAM INTERFACE

INTRODUCTION:

Was first introduced by Dr. Ron Lodewyck-N6EE, founder of Microtronics, in November 1978 to provide radio teletype (tm) and Morse code transmit and receive capabilities on the amateur bands for the TRS-80. Our system arrived November 15, 1978 and was serial number 3, or thereabouts. There appeared to be a hassle over the name of Microtronics, so it was changed a few a months later to Macrotronics and has remained as such ever since.

The M-80 program with a factory built interface printed circuit board sold for $129 over two years ago. Today, the same software and hardware is advertised at $149, so they have held price increases to a modest level on the minimum system. The other options are NOT so modestly priced and are advertised as follows:

CM80 - Same as M80 in cabinet . . . . $ 279.

TM80 - Same as CM80 + active filter RTTY demodulator AFSK with RTTY tuning meter in cabinet . . . . $ 499.

M800 - Adds split screen operation to all models. . . $ 99.

Since Macrotronics would not accept our original system as a trade-in on one of the newer and more expensive systems, we will just have to write about what our $129 investment consists of and how efficiently or poorly, it operates. As usual, CAVEAT EMPTOR or its your own fault. Our observations are based upon using a reasonably well shielded ITT-3021A digitally tuned receiver ($5000 price class) and a Hallicrafters HT-37 single sideband transmitter. The high frequency multi-band 10, 20 & 40 meter trap loaded dipole antenna is in the attic about 50 feet away from the TRS-80 and fed with well shielded RG8/U polyfoam insulated coaxial cable.

The TRS-80 and accessories are fed 110/120 vac power via Electronic Specialists model ISO-1A isolators/filters to decouple most RFI from the power lines. With ALL those WHYFORES and HERETOFORES let's take a brief look at system operation and how well it works for us in our installation.

GENERAL:

Ideally, ANY RTTY or Morse code system should have the station's transmitting/receiving antenna AT LEAST 70 to 80 feet away from the TRS-80 so that radio frequency interference (RFI) radiated by the Model 1 TRS-80 has died down to an acceptable level; i.e., the inverse square law as mentioned earlier in this Chapter. RFI generated by the Model 3 will be

covered in considerable detail in Volume 4 and hopefully RFI will be reduced to the insignificant level to meet the new F.C.C. regulations that go into effect January 1, 1981.

The M-80 software is divided into two segments: a machine code program of approximately 2750 bytes length that is loaded into MEM beginning at 30000 decimal and a BASIC segment of about 7224 bytes length. As such, the program will NOT work with disk unless you have a stable full of cryptographers. They do NOT now sell a disk version for an additional $10, which they initially offered original M-80 purchasers. Macrotronics is now offering their M8000 disk based RTTY program for an additional $150 on top of the M80 or TM80 initial price and requires dual disk drives and a minimum of 32K MEM.

M-80 SYSTEM HARDWARE:

Consists of a printed circuit board measuring 5" by 4 1/2" with a 15 pin male PCB connector on the bottom edge. The transformer is of the 'plug-in' variety and delivers 12 VAC at 500 milliamps. The quality of the PCB, its layout, and the components are FIRST CLASS and a tribute to the professionalism of Ron Lodewyck. All 7 of the integrated circuits plus the Magnecraft keying relay are mounted in sockets and easily removed if necessary, whereas MOST cheapy amateur PCBs would have had the ICs wave soldered directly to the board. In addition to the 7 integrated circuits there are 6 transistors, numerous diodes, 2 LEDS, and an opto-isolator. The circuit was obviously designed with the 'hard core' RTTY enthusiast in mind who uses a surplus TTY machine for hard copy. It is not needed or used by those with a TRS-80 compatible printer.

The M-80 will transmit and 'sort of' receive Morse code (more later), and receive Baudot RTTY, but requires an external frequency shift keyer (FSK) or audio frequency shift keyer (AFSK) to transmit RTTY. Also, our M-80 will neither transmit or receive ASCII RTTY (which the last section of this Chapter is all about).

The heart of the M-80 in both the Morse and Baudot receive modes is an NE-567 phase locked loop (PLL) that is connected to your receiver's speaker output. It may be adjusted in frequency between about 500 and 2500 cycles with a trimmer on the PCB. The 3 db down (1/2 power) bandwidth of the PLL is around 15% of center frequency so that when set at 2000 cycles it would have a passband of plus and minus 150 cycles. This PLL's ONLY job is to tell data bus zero whether a signal is PRESENT or NOT PRESENT at your receiver's speaker terminals. The PLL's 1 or 0 status is output to data bus zero via a 74LS367 and 74LS04 buffer. It is the M-80 program's job to decode it and display it on video.

Since the M-80 cannot transmit Baudot RTTY without a terminal unit, we will only cover the Morse code transmitting aspects. The M-80 offers 3 varieties of transmitter keying including:

1. - 5 VDC negative (grid block) keying
2. + 5 VDC positive keying
3. keying relay - single pole double throw contacts

One of the most fascinating aspects of N6EE's circuit design is that it DOES NOT use the OUT instruction to actuate the keying relay, but rather uses the IN instruction via port 3 to do so. Since no information has to be transfered, just ON or OFF, it works quite well indeed. Address bus lines 0, 1, 2, and 3 are used to feed a 74LS154 4 line to 16 line decoder/ demultiplexer whose bell is 'rung' by the IN instruction. Only the first 5 decoded outputs are used so the other 11 are available for any purpose you may wish. Decoder outputs 0 and 1 drive the CLEAR and PRESET terminals of one of a dual J-K flip-flop, and decoder outputs 3 and 4 drive the CLEAR and PRESET terminals of the other J-K flip-flop. Decoder output 2 actuates the tristate 74LS367 buffer allowing it pass along the status of the PLL to data bus zero. It surely is somewhat different, but it works.

The circuit board also includes an NE-555 audio oscillator to give you an audio side tone via a speaker, if desired.

M-80 MORSE CODE OPERATION:

Is rather straightforward in its simplicity. After loading both the object code and BASIC programs you will be asked:

MORSE OR BAUDOT RTTY (M/B) ?

Assuming you entered an 'M' for Morse, you are off and running. You will then be asked to enter the code speed desired in words per minute, 10 to 100.

Unfortunately, unlike Electric Pencil or the better quality programs, you are NOT given a subcommand table with a menu to allow relatively easy subcommand selection. First you press the CLEAR key and then the following key listed below. You must either memorize or write down this illogical list:

#	=	Start code practice mode
$	=	Change character spacing
%	=	Change code speed
&	=	Create a message (0 through 9)
'	=	Keying N = negative P = pos./relay
(	=	Change to 32 characters per line
)	=	CLS and return to 64 characters/line
0-9	=	Send message number input
R	=	Transfer to RTTY TRANSMIT mode
ENTER	=	Switch to Morse RECEIVE

The program quite naturally sends letter PERFECT Morse code with internationally agreed upon timing ratios of dot = X, dash = 3X, element spacing = X, space between completed characters = 3X, & space between words = 7X.

Additionally, the pause after a numeral 0 - 9 is transmitted is equal to 7X since this a 5 element character

The Morse code RECEIVE mode called by pressing the CLEAR key and then the ENTER key is specified to handle Morse code speeds of "approximately plus or minus 10 words per minute" from that selected during initialization. On our M-80 system, the Morse receive speed MUST be specified within about 2 to 3 words per minute to receive W1AW machine transmitted code at the 15 words per minute rate.

ANY Morse code receive program (including the author's rather slow BASIC Morse program), will be adversely affected by signal fading, interfering signals, static (local or distant) noise, change of code speed, and swing fists (non-standard lengths of dots, dashes, and spaces). Some widely advertised Morse code computer systems, even those in the $2000 category, cannot possibly anticipate lightning noise, man-made noise, SWING FISTS, plus other interference and all the variables that go into decoding a Morse signal. The major points are:

1. Morse code receive systems, most ANY kind, will work quite well under perfect conditions when receiving machine or Iambic keyer generated IDEALLY timed Morse code.

2. Under any OTHER conditions, it is the author's opinion that copying Morse with the human ear will invariably BEAT most any variety of computerized decoding system in the range of 5 to 40 words per minute. If you want to GO faster, use ASCII or Baudot RRTY. Morse WAS great for reporting Custer's last stand via telegraph, adjusting artillery shots from balloons in the Civil War, and for the Titanic's SOS, but that was a long, long time ago.

3. The ONLY exception to the above is WHEN a 'sort of' full-duplex program is utilized that allows the receiving station to transmit back to the originating station either each letter received, each word received, or the full message, THUS allowing EXACT checking of each letter received. IF a wrong letter is detected, it is then re-transmitted, checked again, and the process continued until the CORRECT letter, word, or message is received. This is a system used by a number of Scandanavian ship to shore RTTY communication systems.

All the foregoing is not meant to be a criticism of N6EE's hardware or software. It just happens to be a 'fact of life,' whether you like it or not.

M-80 RADIO TELETYPE (tm) MODE:

We rate this segment of the program 'FAIR TO GOOD.' Considering the fact that N6EE spent a year writing it, it should be. After initialization and after answering the MORSE OR BAUDOT RTTY ? with a 'B', you will find yourself in the RTTY TRANSMIT mode with the Baud rate initialized at 60 words per minute.

As in the Morse code mode, N6EE forgot to use a subcommand table to allow the user to easily modify the program. The following must be either memorized or written down. The CLEAR key is first pressed and the following key then pressed to:

#	=	Auto Morse ID from message zero
$	=	Reverse mark and space tones
%	=	Change Baud rate: 60, 66, 75, 100 WPM
&	=	Create message 0 to 9: 255 char. max
'	=	Select unshift on space - turn it off
(	=	Change video to 32 characters/line
)	=	CLS and resume 64 characters/line
0 - 9	=	Transmit message number N
M	=	Go to Morse TRANSMIT mode
ENTER	=	Go to RTTY RECEIVE mode

M-80 RTTY TERMINAL UNITS AND AFSK GENERATORS:

As previously mentioned, the M-80 requires a terminal unit to generate the proper frequency shift on most all amateur SSB transmitters. The exceptions are the relatively NEW models from ICOM (which we favor), Kenwood, and Yaseu which may be directly keyed from the M80 keying relay to obtain narrow band (170 cycle) frequency shift output. There are many terminal units on the market with widely varying prices. In our opinion you can probably build a better one from scratch or from a kit than you can buy. Ham Radio, 73, and QST magazines all have advertisements for most all of them. Our favorite RTTY demodulator is the DIGIRATT dual phase locked loop unit in kit form available for $35 ppd from Circuit Board Specialists, P.O. Box 969, Pueblo, Colorado 81002. It is one of the finest kits we have seen and should sell for $100. For those who wish to dig deeper, this excellent dual PLL demodulator with two stage active filter is described by its creator, John Loughmiller-KB9AT, in the October 1978 issue of Ham Radio Magazine.

The foregoing RTTY demodulator would replace the single NE-567 PLL in the M-80 which is really only a 'toy' for beginners.

For generating the audio frequency shift keying (AFSK) tones, 170 cycles shift for narrow band Baudot RTTY and 850 cycles shift for wide band Baudot RTTY, crystal control is the only way to go. Though we have not seen it, we suspect that the Macrotronics XTL-1 unit at $79.95 is probably as good as any factory built unit on the market. For 200 cycle frequency shift ASCII RTTY, the systems described in sections 1 and 3 of this Chapter should serve as starters.

The only significant criticism we have regarding the M-80 RTTY transmit program is that N6EE forgot to include a characters per line counter in the program. Most all old surplus TTY machines have 72 characters per line and require BOTH a carriage return AND line feed to begin the next line. In the M-80 program you MUST manually hit the ENTER key to effect

BOTH a carriage return and line feed IF the TTY machine on the receiving end is a standard unit. One way to get around this oversight is to send only 63 characters per line on your TRS-80 video display and manually hit the ENTER key at the end of each line. Each prepared message, 0 through 9, should be limited to 62 characters PER LINE and BOTH started and ended with an ENTER. Each message must not exceed 255 chars. total.

M-80 RTTY RECEIVE MODE:

Is quite reliable, once you 'tame' the Model 1's radio frequency interference problem. We manage to copy W1AW RTTY bulletins without difficulty on 14.095 MHz using only the simple M-80 NE-567 PLL demodulator that comes with the circuit board. For any weaker signals, we recommend the DIGIRATT dual phase locked loop demodulator that was previously mentioned. Its extremely clever design and the two stage active filter allow one to receive RTTY signals where selective fading has totally wiped out one or the other frequency shifted tones. It is easily adjusted to either 170 cycle or 200 cycle frequency shift, or may be switched if you are careful to use a STABLE 20 turn trim potentiometer to tune the additional frequency.

CONCLUSION OF SECTION TWO:

We MAY have been somewhat harsh in our criticism of the M-80 Macrotronics Morse/RTTY program. IF so, it was intentional. We understand that their new M8000 disk based RTTY program does NOT include Morse code. IF so, that was GOOD judgement on N6EE's part as NO really foolproof (literally and figuratively) Morse RECEIVE program can be written that works reliably under all circumstances unless the laws of nature are changed.

This is not meant to intimate that Morse code should be ignored or forsaken by the radio amateur. Just the opposite is intended as there are times when ONLY Morse signals will get through. Morse has it excellent points, but they are NOT part of any TRS-80 program EXCEPT for transmission.

The M-80 system, taken as a whole, is certainly worth the current $149 price. The M-80 circuit board is of excellent quality and the program is rated fair to good. Let us hope that N6EE sees fit to improve the program even further which should be an easy task for one with his outstanding ability.

REFERENCE:

Macrotronics, Incorporated
Dr. Ron Lodewyck-President Mrs. Elizabeth Lodewyck-V.P.
1125 North Golden State Boulevard - Suite G
Turlock, California 95380

PHONE: (209) 667-2888 and (209) 634-8888

SECTION III - CHAPTER 10

ASCII TRANSMITTING AND RECEIVING SYSTEM

INTRODUCTION:

Since the recent approval of ASCII code for RTTY transmissions by the F.C.C. there has been a surprising LACK of interest in this subject by the amateur fraternity even though the American Radio Relay League amateur station W1AW has been sending RRTY bulletins daily in BOTH Baudot and ASCII since July 1, 1980. Our best guesses for this apparent lack of enthusiasm are two-fold:

1. The tens of thousands of surplus TTY machines out there in amateur radio land are Baudot code machines NOT ASCII.

2. The vast majority of microcomputers, the TRS-80s, the Apples, and the Pets, built and delivered prior to January 1, 1981 are about the worst radio frequency noise generators next to military ECM (electronic counter-measures) systems that one can imagine.

Undoubtedly the two foregoing facts are the major contributors to the lack of unbridled, worldwide, amateur radio enthusiasm for ASCII instead of Baudot code. Another contributing, but certainly minor factor may be lack of standardization of ASCII transmissions on the high frequency amateur bands. These would include: frequency shift (170 or 200 cycles), bits transmitted (7 or 8 bit mode), parity or no parity bit, and Baud rate (110 or 300).

Since the ARRL bulletin station, W1AW has been the leader in getting on the air with ASCII transmissions on a daily basis, following is the W1AW schedule and frequencies they use for transmitting the bulletins of interest to radio amateurs. This schedule is in eastern standard time and covers the period October 26, 1980 through April 25, 1981. During the daylight saving time season, the times are moved forward one hour, so are the same EXCEPT in EDST instead of EST.

W1AW RTTY BULLETINS - FIRST IN BAUDOT - SECOND IN ASCII

MHz 1.835, 7.095, 14.095, 21.095, 28.080, 50.080, 147.555

DAY	M	T	W	T	F	S	S
TIME	11AM	11AM	11AM	11AM	11AM	-	-
	6PM	6PM	6PM	6PM	6PM	6PM	6PM
	9PM	9PM	9PM	9PM	9PM	9PM	9PM
	12PM	12PM	12PM	12PM	12PM	12PM	12PM

The bulletins are transmitted at 110 Baud, 7 bits data + 1 space bit, and 2 stop bits. The space bit = parity ON as far as our programming the TRS-80 RS-232C adaptor is concerned.

Setting up our RS-232C interface to receive W1AW's ASCII transmissions would look like this:

```
Master reset   =  OUT232,255   or OUT (232),A   (any value ok)
110 Baud rate =  OUT233,34    or OUT (233),A   (A = 22H)
```

UART CONTROL REGISTER AND HANDSHAKE CONVENTIONS FOR W1AW

```
BIT 7 (MSB)    =  0 for odd & 1 for even parity (not used)
BIT 6 (NMSB)   =  0 for 7 bit word
BIT 5 (NMSB)   =  1 for 7 bit word
BIT 4 (NMSB)   =  1 for 2 stop bits
BIT 3 (NMSB)   =  0 for parity enable (SPACE bit used by W1AW)
BIT 2 (NMSB)   =  1 for enable transmit data
BIT 1 (NMSB)   =  0 for request to send
BIT 0 (LSB)    =  1 for data terminal ready
```

00110101 BINARY = 53 DECIMAL or 035 HEX FOR PORT 234

The next step would be to:

```
SET PORT 234  =  OUT233,53  or OUT (234),A  (with A = 035H)
```

Now let' try writing a mini-BASIC program JUST to copy W1AW ASCII RTTY transmissions with our TRS-80. For openers, we will try using our MODEM held close to our receiver's speaker and tune our receiver's BFO (beat frequency oscillator) so that our 200 cycle shift MODEM may possibly copy the signal. After all, 170 cycle shift is only 15 percent less than 200 cycles. We will try receiving W1AW's ASCII RTTY on the 20 meter amateur band around 14.095 MHz. IF we set up our receiver for upper sideband and adjust the BFO (beat frequency oscillator) carefully, then the audio tones from the receiver's speaker should be in the proper order with about a 2225 cycles audio note = mark, and 2225 - 170 = 2055 cycles note for space. We can use the 'READY' light on our MODEM as a 'sort of' frequency meter by adjusting the BFO until the 'READY' light comes on. Center the BFO frequency with the light & tone.

Let's see if it will work with this simple BASIC program:

```
10 ' W4UCH/2 MINI-BASIC PROGRAM TO COPY W1AW ASCII RTTY
20 '
30 OUT232,255:OUT233,34:OUT234,181
40 IFINP(234)<128GOTO40ELSEIFINP(235)<32GOTO40
50 PRINTCHR$(INP(235));:GOTO40
```

We will have our MODEM set with right hand switch in the 'F' position and left hand switch in 'O' position. The READY light & tone (carrier detect) on the MODEM will come 'ON' when we have the audio output from our receiver's speaker adjusted to around 2225 cycles. Try whistling the note 'C' above middle 'C'. The 2nd harmonic will light it briefly. The extact frequency we want, adjusted with our receiver's BFO, will be indicated when the above mini-program begins printing out the daily ARRL bulletin to amateurs on the video display.

SONOFAGUN, IT WORKS ! ! ! SINCE IT WAS SO EASY, WHY DIDN'T RADIO SHACK TELL US WE COULD RECEIVE RTTY WITH THEIR MODEM ? ?

That is a good question, Gridley. To avoid future law suits, perhaps I had best not answer. Let us be generous and assume that it was just plain 'ole STUPIDITY rather than any kind of malice aforethought. Radio Shack has many radio amateurs working for them at Tandy Center. Most try to do a good job for Radio Shack merchandising TRS-80s, BUT have probably never considered such a complex program as the little three liner on the last page.

Let's take a look at part of a typical ARRL bulletin to radio amateurs that was printed out on video by our Model 1 TRS-80 on November 14, 1980. The frequency was 14,091.3 kHz at 11:00 AM eastern standard time. It used the standard TRS-80 RS-232C interface board and standard MODEM about two inches away from our receiver's speaker. The ONLY program was the mini-BASIC one on the last page. All three lines of it. The W1AW bulletin was output to video and simultaneously taped for later print out. Most of the QRM by a W1 was edited out.

W1AW BULLETIN TO RADIO AMATEURS NOVEMBER 14, 1980.

U* U* U* U* U* U* U* U* U* U* U* U* U* U* U*
DX 44 FOLLOWS
QST DE W1AW
HR DX BULLETIN NR 44
FROM ARRL HEADQUARTERS NEWINGTON CT
NOVEMBER 14, 1980
TO ALL RADIO AMATEURS

THANKS TO THE SOUTHERN NEW ENGLAND DX ASSOCIATION FOR THE FOR THE FOLLOWING DX INFORMATION.

CHATHAM ISLAND.
ZL1AMO FROM 20 NOVEMBER UNTIL 3 DECEMBER.

QATAR.
A7XD SHOWS UP FRIDAYS AT 0200Z ON 7090 OR 3790.

CEUTA (Y MELILLA, SPANISH)
EA9XXXX ACTIVE DAILY AT 0600Z ON 3795, LOOKING FOR KH6 TO COMPLETE 80 METER WAS.

KERMADEC ISLAND.
ZL3MA/K TO ARRIVE 22 NOVEMBER FOR TWO WEEKS, EMPHASIZING EUROPEAN QSOS.

JUAN DE NOVA.
QSLS FROM THE RECENT DX EXPEDITION SHOULD ARRIVE SHORTLY ACCORDING TO DL1YD.

PACIFIC.
DL1VU AND DJ7FX WILL BEGIN 22 NOVEMBER STARTING AT A35 AND MOVING ON TO ZK2 ZK1 FO8 5W1 KH8 3D2 AND FK ENDING AROUND

1 MARCH. QSL TO DL2RM.

TOGO.
5V7HL KEEPS A SKED ON SUNDAYS ON 21355 AT 2100Z.

VATICAN CITY.
I0DUD SHOULD BE ON NOW FROM HV3SJ.

MALAGASY REPUBLIC.
LOOK FOR 5R8AL ON 21 NOVEMBER AT 1700Z ON 21290. MAY BE A LIST OPERATION. (work only their friends/financial supporters)

REPUBLIC OF GUINEA.
IAN, VK4NIC/3X WAS ON BRIEFLY LAST WEEK PROMISING COMPLETE DOCUMENTATION LATER. SEVERAL HUNDRED QSOS WERE MADE BUT HIS WEAK SIGNALS AND LOW ANTENNAS MADE THINGS ROUGH. EXPECT HIM TO RETURN IN ABOUT THREE WEEKS AND TO STAY FOR SIX MONTHS.

EL2CAM MAY LEAD A GROUP TO CONAKRY DECEMBER 31 AND JANUARY 1 ACCORDING TO THE RUMOR MILL.

JOHNSTON ISLAND.
KH6GB/KH3 IS ON NOW, WATCH 15 AND 10 METER PHONE.

NOTE:
The very few XXXXs above were caused by interference (QRM) from a thoughtless W1 amateur radio station who was working Baudot RTTY almost on top of W1AW's ASCII RTTY frequency. Even amateur radio has its careless and/or malicious fringe who delight in QRMing ASCII transmissions. Fortunately, this squirrel (nut) was almost totally ineffective.

- -

Well now Gridley, what do you thing of that DX (long distance) RARE countries status report from the American Radio Relay League's station, W1AW, decoded and printed out by our little TRS-80?

EXCEPT FOR THE VATICAN, I'VE NEVER EVEN HEARD OF THEM ! ! !

Do not feel badly, Gridley. Most of us have never heard of very many of them either, though we did work a bit of DX ourselves out of the MARS station on Johnston Island in 1950/1951. The DX fraternity (those who get their kicks out of working every country/island/atoll or sand-spit) in the world are a real different breed of radio amateur.

You should remember that old fashioned standard teletype (tm) machines have exactly 72 characters to the line compared to our TRS-80's 64 characters per line. If you plan to do any serious teletype work, either Baudot or ASCII, a characters per line counter (like 40A6H in MEM) is a virtual necessity.

THAT WAS A RATHER IMPRESSIVE DEMONSTRATION, PROFESSOR. WHAT DO YOU PLAN FOR AN ENCORE ? ? ?

Well Gridley, not very much as this Chapter has run far too long already. We will save the neat little DIGIRATT PLL2 demodulator and the details for a homebrew RTTY modulator with both 170 cycle and 200 cycle audio frequency shift keying for Volume 4. The fact that the standard Radio Shack MODEM and RS-232C interface worked as well as they did for receiving amateur band ASCII radio teletype (tm), probably surprised us as much as it did Radio Shack. The additional fact that it would work with a truly simple BASIC program, gave us the greatest pleasure of all.

Here is another extremely simple BASIC program that will allow you to intialize the RS-232C/UART adaptor at 110 or 300 baud, even or odd parity, 7 or 8 bit word length, and parity ON or OFF. Since most 110 Baud RTTY uses 2 stop bits and 300 Baud 1 stop bit, this is automatically set by the program in line 100. The ELSE IF INP(235) <32 GOTO 140 in line 140 merely inhibits any carriage returns and line feeds to save paper. Leave it out if you wish.

```
10 ' W4UCH/2 MINI-BASIC PGM TO COPY HF ASCII RTTY WITH MODEM
20 '
30 OUT232,255
40 INPUT"110 OR 300 BAUD RATE ";X
50 IFX=110THENOUT233,34ELSEOUT233,85
60 INPUT"EVEN OR ODD PARITY (1/0) ";A
70 IFA=1THENA=128ELSEA=0
80 INPUT"7 OR 8 BIT WORD LENGTH ";B
90 IFB=7THENB=32ELSEB=96
100 IFX=110THENC=16ELSEC=0
110 INPUT"PARITY 'OFF' OR 'ON' (1/0) ";D
120 IFD=1THEND=8ELSED=0
130 E=5:Y=A+B+C+D+E:OUT234,Y
140 IFINP(234)<128THENGOTO140ELSEIFINP(235)<32GOTO140
150 PRINTCHR$(INP(235));:GOTO140
```

CONCLUSIONS:

Radio teletype (tm) using the Model 1 TRS-80 is about as simple as falling off a log. By adding a few lines to the above program you may OUTPUT from the keyboard via the INKEY$ function after testing BIT 6 of PORT 235 for the UART transmit holding register EMPTY. Then, using OUT235,xxx to send the character from the MODEM, it may be picked up by your amateur station's microphone and transmitted on ANY of the HF, VHF, UHF or microwave bands you wish. It will work just as well with a Collins or Heathkit rig on 160 meters as it will with a Gunnplexer on the 10.250 gigahertz microwave frequency.

Sophisticated RTTY terminal units with all their goodies and a meter or CRT to assist in tuning are NICE to have, but are certainly NOT necessary IF you have a TRS-80 to do the job.

- CHAPTER 11 -

SELF TEST QUESTIONS FOR CHAPTER 1

1. What was the Indian Princess' name? Her calling card?

2. Where did the calling card come from? See the cover to Harv Pennington's excellent book, "TRS-80 Disk & Other Mysteries."

3. How many instructions in the Z-80's instruction set?

4. What are the Level II BASIC decimal function codes for: AND, [, /, *, -, +, STEP, NOT, THEN, INKEY$, MEM ?

5. IF the first byte of the object code is 203, 221, 237, or 253 how does the program handle it on page 16?

6. Does HEX have anything good going for it when used with the TRS-80?

7. Who's gonna spend $10 and enter a crazy contest like the one on page 25?

8. Which is the FASTEST disassembler program beginning on page 29? STRING manipulation or INT decoding?

9. Approximately how many bytes are save by the programs beginning on page 29 compared to the program on page 12?

10. Is the shortest program (MEM requirement) usually the fastest program?

11. Are there any other ways to switch LPRINT for PRINT, or vice versa, than that shown on page 33?

12. What does ECHO video on LPRINT mean?

13. How can one do it?

14. Is it an easy task to disassemble NEWDOS+'s disassembler?

15. If NO, why not?

16. How much MEM does the NEWDOS+ disassembler use?

17. Why did we NOT include the IY index register instructions and most of the SET and RES instructions in this disassembler?

18. Is the IY index register used by NEWDOS+?

19. How would you ADD the IY index register object codes and instuctions to this disassembler program?

20. Is there ANYTHING complex or difficult to understand about writing a disassembler or Editor/Assembler program?

- SELF TEST QUESTIONS FOR CHAPTER 2 -

1. Who WON first prize in the 'Fastest Foot Race Contest' ?
2. Is he a professional program author ?
3. Was he a professional fighter pilot ?
4. Who does he write programs for now ?
5. What is the name of his newest program ?
6. How much does it cost ?
7. Where can you purchase it ?
8. How much faster was the first place winner than the second place winner ?
9. Can you write a FASTER disassembler, staying within the contest rules ?
10. Were 5 out of the 10 fastest programs written by professionals?
11. Who qualifies as, 'A Professional' ?
12. Who won the 'MOST UNIQUE' program prize ?
13. Is he a professional ?
14. Did you learn ANYTHING from studying the 'MOST UNIQUE' prize winning program ?
15. Could you write a more attractively presented program than the 'MOST UNIQUE' program prize winner ?
16. How many bytes used in the first prize winning program ?
17. How many bytes used in the 'MOST UNIQUE' prize winning program ?
18. Is the SHORTEST (bytes used) program usually the FASTEST ? Is the LONGEST (bytes used) program usually the SLOWEST ?
19. Who won the FIRST TEN places in the contest ?
20. Would RICHCRAFT ever sponsor another contest ?

- SELF TEST QUESTIONS FOR CHAPTER 3 -

1. How are we simulating a FIFO in this Chapter? Is our FIFO output serial or parallel? How many bits wide?
2. What is the bandwidth of a typical Bell System telephone line?
3. How can we change parallel data to serial? Vice versa?
4. Which is the most cost effective solution to converting parallel data to serial or vice versa: a software program or UART chip?
5. Why DEFINT in line 30 of this Chapter's BASIC program?
6. Why input a few lines of text before hitting right-arrow?
7. What is one of the slowest microcomputer peripherals?
8. How would YOU change the BASIC demo program to CLS after printing the last video line and wait for you to enter another few lines before resuming LPRINT?
9. How does a separate FIFO chip tell you it is full?
10. Why do we use the IY index register in the second pgm.?
11. What is the difference beween RIGHT ARROW and SHIFT-RIGHT ARROW?
12. What is the difference in output between CALL 032AH with 409CH = +1 and CALL 003BH?
13. Could alternate register AF' have been used as a counter instead of PRINT, CARRET, or LINES in the 2nd program?
14. How about alternate register IY' as a counter too?
15. About how many 'open' (= NOP) bytes between MEM locations 17129 and 26810 decimal?
16. May these 'open' bytes be used for assembly language programs?
17. Where is Fagginsylvania? Who is it named after?
18. How could one frequency hop around a band to avoid interception? Name 2 ways. Are they both TOTALLY secure?
19. Who helped win World War II's "War of The Atlantic With Submarines." Company name? Scientist's name?
20. Where do un-used bytes go after scrolling off of your video display? Are you sure?

- SELF TEST QUESTIONS FOR CHAPTER 4 -

1. To install a normally OPEN pushbutton external interrupt switch to the TRS-80, to which leads to the rear connector would you solder the two switch wires ?
2. Why doesn't the text use interrupt modes zero or two ?
3. What function does the Z-80 IFF1 flip-flop perform ?
4. What function does the Z-80 IFF2 flip-flop perform ?
5. What do some other (not TRS-80) microcomputers use the non-maskable interrupt for ?
6. Does the EI, enable interrupt instruction, enable interrupts immediately ? If NOT, why not ?
7. Where is the program counter stored during an interrupt ?
8. Which is the most versatile interrupt mode ? Can we use it with our TRS-80 ? Why not ?
9. Can an interrupt subroutine be written in BASIC ?
10. How many fingers does Gridley have on each hand ?
11. What MEM location does our assembler use to 'stuff' an assembly language programs beginning address when we use END and the beginning address of a source code program ?
12. Is the above function ALWAYS reliable ?
13. Why do we SAVE everything in Page 4-13's program ?
14. How do we SAVE everything in Page 4-15's program ?
15. What price do we pay ?
16. Why do we PUSH & POP the IX and IY registers in page 4-15's program ?
17. Name two VERY useful applications of interrupts for the TRS-80 ?
18. Did the author ever write an IM2 program for the TRS-80 ?
19. Did it work ?
20. WHO wrote the easy questions and simple answers for this Chapter ?

- SELF TEST QUESTIONS FOR CHAPTER 5 -

1. What kind of TTL chip is Z4 in Figure 5-1 and WHAT does it accomplish ?

2. What kind of TTL chip is Z59 in Figure 5-1 and WHAT does it accomplish ?

3. What function does Z52 and Z54 perform in Figure 5-1 ?

4. How does flip-flop Z25 determine whether it is an OUT or IN instruction ?

5. What function does hex buffer Z44 perform during CLOAD ?

6. How does one use Port 255 to input a signal without cobbling up any printed circuit boards ?

7. How many relays could we control JUST using port 255's decoder and ALL the DATA BUS lines ? Careful, Gridley.

8. Do we recommend opening up the keyboard and cutting traces to anyone with a razor knife and soldering iron ?

9. Why do we recommend the Telesis VAR/80 Interface for the TRS-80 ?

10. How many relay controlled outputs in the VAR/80 ? Which ones ? LED "relay closed" lights for the relays ?

11. What type of relays are used ? What is their maximum voltage and current rating ?

12. How many individual outputs total ? How many inputs ?

13. How many 'OPTO-COUPLER' isolatated inputs ? What rating ?

14. How many address lines are decoded ? Is it a disadvantage?

15. How many OUT or INP port addresses will it answer to ?

16. Can you change the port address decoding ?

17. Who is 'Mr. V. A. R.' ?

18. What does the Z-80 instruction RRCA accomplish ?

19. Can the VAR/80 be easily expanded to utilize MORE than the eight DBO outputs ?

20. How could you drive a VAR/80 input from 120 volts ac WITHOUT using a relay ? Is it safe ? Do we recommend using it ?

- SELF TEST QUESTIONS FOR CHAPTER 6 -

1. Who manufactured one of the first A/D converters circa 1955 ? Price ? Weight ? Solid-state ?

2. Define time constant (seconds) for R = ohms & C = farads ?

3. Who invented natural logarithms ? Where ? When ?

4. What type of A/D converter is our homebrew unit ?

5. What does line 20 in page 6-8's program accomplish ?

6. What are the three major errors of A/D converters ?

7. What is the major component contributing to errors in our homebrew A/D converter ? Why ?

8. What are the two major disadvantages of the servo-counter tracking type A/D converter ?

9. What type of A/D converter is the 'fastest' gun in town ?

10. Even at today's mass market prices, are they expensive ?

11. Which type of A/D converter is the most 'cost effective' variety, today ? Who is a leading manufacturer ? Price ?

12. What are the major differences between the ADC0808 A/D conversion SYSTEM and a stand alone A/D converter ?

13. What are the major differences between the ADC0816 A/D conversion system and the ADC0808 conversion system ?

14. How can one obtain a 10% discount off the list price of the Alpha Products 'Analog 80' ?

15. Will the input impedance of the Analog 80 cause erroneous conversion readings for high impedance sources ? What is the Analog 80's input impedance ?

16. What MUST one do with the Analog 80 to run page 6-23's program ? Why ?

17. Are the labels ZERO through SEVEN necessary for the program on page 6-22 to function properly ? Why ?

18. How would one attach 2, 3, or 4 Analog 80's to the TRS-80 ?

19. Why do we use JP 114 instead of JP 1A19H to return to BASIC in page 6-24's program ?

20. Why do we recommend the Alpha Products Analog 80 system rather than some of the considerably CHEAPER A/D converters on the market today ?

- SELF TEST QUESTIONS FOR CHAPTER 7 -

1. How would one rate the dielectric absorption and insulation resistance of a electrolytic capacitor ?
2. Can a Leyden jar be used as a high impedance voltmeter ?
3. Why does line 90 in page 7-5's program delay 1 minute ? Would 10 minutes be even better ?
4. Who built the first 8 bit D/A converter utilizing multiple monolithic integrated circuits ? When ?
5. Most all contemporary D/A converters are _______ types.
6. How is R, 2R, 4R, 8R, 16R, 32R, 64R & 128R used ?
7. What are the primary advantages of this technique ? List three advantages.
8. What is the primary advantage of a 'Deglitched D/A Converter ?
9. About 'how much' parts cost at Radio Shack for the integrated circuits shown in figure 7-6 ?
10. Is the National Semiconductor DAC0808 D/A converter expensive ? How much does it cost ?
11. Do you really need an accurate digital voltmeter to TEST the DAC0808 ?
12. What is the MOST common problem in amateur construction projects ?
13. What is the MOST difficult/tedious part of building the DAC0808/LF351 D/A converter ? Is it really difficult ?
14. Will most ANY 12 volt ac transformer be adequate for the - 15 vdc regulated power supply ? Why ?
15. Why is the accuracy of the feedback resistor on the LF351 opamp shown in figure 7-8 so important ?
16. Are there any modestly priced D/A factory built converters for the TRS-80 available today ?
17. Count from 5 to 8 in FORTH, Gridley ?
18. What are the advantages of the DAC0800 over the DAC0808 ?
19. Is it more costly than the DAC0808 ? How much ?
20. What is the best (that we know of) D/A & A/D handbook ?

CHAPTER 8

NO SELF TEST QUESTIONS AS THE PROGRAM IS PRESENTED FOR INFORMATION ONLY

- SELF TEST QUESTIONS FOR CHAPTER 9 -

1. Does the author object to program swapping on a freebie basis ? Unauthorized selling or renting copies of copyrighted programs ?
2. Will most bulletin board systems include PROGRAM transfer in a few months ? Is that GOOD or BAD ? How is it done ?
3. For 300 Baud rate bulletin boards what is the minimum equipment needed ?
4. Can one substitute software for hardware at 300 Baud ? If not, why not ?
5. Does the author recommend using the RS-232C sense switches ? Why not ?
6. Who prepared the Electronic Industries Assoc. RS-232C specifications ? Where can you obtain them ?
7. Is it relatively inexpensive to set up your own central station bulletin board system ?
8. What is the minimum equipment required to set up your own central station bulletin board system ?
9. To which port does one send an OUT to reset the RS-232C master latch ? Must it be done every time ? Must a special value be output to effect a reset?
10. To which port does one OUT to set the Baud rate ? What value is OUT for 110 Baud ? For 300 Baud ?
11. To which port does one OUT to set even-odd parity, word length, stop bit or bits, and parity ON or OFF ?
12. To which port does one IN or INP to determine receive byte READY, byte TRANSMITTED status, overrun error, framing error, and parity error ?
13. Which port is used to TRANSMIT a byte ? To RECEIVE a converted byte NOW in parallel format ?
14. What value do you OUT to obtain: even parity, a 7 bit word, 1 stop bit, and parity ON ? Same for an 8 bit word ?
15. After dialing up a bulletin board on the phone, how long do you have to put your telephone set onto the MODEM ?
16. What position should the MODEM switches be in when working a bulletin board system ?
17. How may Chapter 9's program be tested without dialing up a bulletin board system ?
18. Who is the famous W6TNS that chapter 9's program is dedicated to ?
19. What does the OR A instruction in line 470 of this Chapter's program on page 9-9 accomplish ? Could it just as well be done with the CP instruction ?
20. Will this program allow the user to TRANSMIT while there are incoming bytes from the bulletin board ? If not, what would be necessary to do so ?

- SELF TEST QUESTIONS FOR CHAPTER 10 -

1. Who invented the first multiplexed telegraph system ?
2. Who invented the first teletype machine ? Financed it ?

3. Who bought the WHOLE U.S. teletype business ? When ?
4. Who invented the Baudot code ?

5. Who is Wayne Green ?
6. What frequencies do the amateur 6 & 2 meter bands cover ?

7. What is the MAJOR difference betweem 1/2 and full duplex?
8. Why are we using the 6 & 2 meter bands ?

9. Which bit is transmitted FIRST by the UART ?
10. How is a single sideband signal generated ?

11. Is TRS-80 radio frequency interference SEVERE at VHF ?
12. How does one minimize it ?

13. Where does one buy a cheapy USED communications receiver?
14. Who manufactures low cost/high quality 6M/2M converters ?

15. Where is a good place to puchase 6M/2M antennas ?
16. What does the author think of Radio Shack manual 26-1145?

17. Who wrote it ?
18. How many stop bits are used for 110 Baud RTTY ? Length ?

19. Will a BASIC program actually work at 110 Baud ?
20. What do we have to tell our program to TRANSMIT a program?

21. What do we have to tell our program to RECEIVE a program?
22. Could you write a SIMPLER program to transfer a program ?

23. Is the text TOTALLY self explanatory regarding this pgm.?
24. Is section II fair to Macrotronics ? - Grudge ?

25. What is BEST about the M80 system ?
26. Is the M80 Morse receive program better than the author's?

27. If so, why? Would you purchase an IMPROVED pgm. from them?
28. Will you report on the IMPROVED program in Volume 4 ?

29. Would you recommend the M80 system to a friend ?
30. WHY no questions on Section 3 ?

31. Is Volume 3 as LONG as you planned it ?
32. Why did you raise the price of Volume 3 by 20 percent ?

33. When will Volume 4 be printed ? How much will it cost ?
34. Can I buy it cheaper IF I reserve a copy now ?

- CHAPTER 12 -

ANSWERS TO SELF TEST QUESTIONS FOR CHAPTER 1

1. WAH-MEE-DAH as in Wahmeda Industrial Park. Bat manure.

2. Harv Pennington's TRS-80 is a belfry full of bats.

3. Surprise: there are 696. See EDTASM User Instruction Manual pages 22 & 23, LD A,R = ED5FH and LD R,A = ED4FH. These instructions/object codes were inadvertently left out of of the summary lists on pages 117 and 121/122 of our 1978 users manual. They MAY be in later printings.

4. 210, 209, 208, 207, 206, 205, 204, 203, 202, 201, 200

5. Shoots us off to lines 3999, 5000, 6000, or 7000 for the appropriate multi-byte object code.

6. Of course, HEX has many good features too. HEX object codes are SOMETIMES easier to follow and most ALWAYS take less space to PRINT......especially 4 byte object codes.

7. Any answer acceptable. We will just have to wait and see.

8. Time both of them and find out for yourself.

9. Approximately 1600 bytes if you only changed lines 100 through 2550. Over 4800 IF you included ALL Z-80 object codes/instructions and utilized the same scheme for all 203, 221, 237 and 253 subroutines.

10. VERY, VERY SELDOM when written in either BASIC or assembly language. Remember: JR's take 33% less MEM than JP's, but require 20% MORE execution time.

11. There sure are many other ways to skin this cat. Try switching the line printer and video display control blocks at MEM locations 16421 and 16413, respectively. See Level II BASIC Reference Manual, page D/1. For proper operation, it is NOT as easy as it looks.

12. Quite simply, intercepting the next character to be LPRINTed or displayed on video, and echoing it to the other before or after it is output.

13. Here is ONE of the myriad ways to do it. Substitute the address in MEM of a brief assembly subroutine for the video display control block's driver address at 16414 and 16415 in MEM. Have this subroutine output the next character to be printed to the line printer and then JP to 1112 decimal/0458H to output it on video. Do not forget to set up your own characters per line, line counter for carriage returns, OR derive it from cursor location. Do not overlook SPOOLING which is covered in a later Chapter; i.e., time sharing without interrupts.

- ANSWERS TO SELF-TEST QUESTIONS FOR CHAPTER 1 CONTD. -

14. NO. This is a real tangled web for hotdog unweavers. You had best carry along a ball of string to find your way home.

15. The Bobsey twins (Cliff and his brother at Apparat) who wrote this disassembler have a substantial proprietary interest in safeguarding their property. As such, this relocatable program jumps around somewhat more than usual. HINT: make a spare disk and use your SUPERZAP to make it stand still. Then disassemble it at your leisure beginning with the program instructions which are close to the END of the program and then work backwards. This is a great pastime for rainy Saturdays and Sundays after your local football team has been eliminated from the playoffs.

16. 26 X 256 = approximately 6656 bytes. This is certainly a VERY long machine language program, but it is definitely the very BEST one on the market today.

17. As previously mentioned, we wished to leave the reader with a 16K MEM TRS-80 some working room. If you have MEM to spare, by all means finish the program as it can be fully implemented in about an hour or so.

18. Fundamentally, no. It is not used by DOS+ or disk BASIC, BUT it is used extensively by the NEWDOS+ disassembler and a number of other programs, so certainly add it on IF this is your ONLY disassembler.

19. Quite simply; add both 7000+ and 9000+ line numbers EXACTLY the same as lines 5000+ and 8000+, except change all the IX's to IY's, and have line 7203 read:

```
IFE=203GOTO9006
```

20. Absolutely nothing complex or difficult IF you take a small bite at a time. Either one may be written in BASIC using simple IF-THEN statements and a few ancillary subroutines to manipulate strings and/or PEEK & POKE the correct data into/from MEM. Many students of the subject write Editor/Assemblers as good as, or better than, the Radio Shack/Apparat version. Send us a copy for the archives when you are finished with this interesting challenge. You SHOULD have enough 'know how' and knowledge of the subject by now, to readily do so.

- ANSWERS TO SELF TEST QUESTIONS FOR CHAPTER 2 -

1. Colonel Charles D. House, USAF (Retd)

2. Yes.

3. Yes.

4. Computer Information Exchange, Box 159, San Luis Rey, California 92068

5. SuperPIMS - data base management system - "The most powerful and versatile sequentially accessed data base system available today."

6. $16.95 postpaid.

7. Computer Information Exchange, Box 159, San Luis Rey, California 92608 phone: (714) 757-4849

8. One minute and six seconds.

9. We doubt it. If you had, you would be richer, today.

10. Yes.

11. ANYONE who writes an original program GOOD enough to be sold in volume AND whose purchasers would recommend it to a friend (not enemy).

12. Mr. H. Woods Martin.

13. No, but he probably will be shortly.

14. LOTS ! ! ! Do yourself a favor and study BOTH the layout and programming technique.

15. NO, but we wish we could.

16. 12456 bytes.

17. 16724 bytes.

18. NEVER, well hardly ever - NEVER, well hardly ever.

19. Charles D. House, Joel Mick, H. Woods Martin, James M. Curran, Charles T. Hart, John Blair/John Coble, John W. Blattner, Hubert S. Howe Jr., Gregory Gilley, Jon. M. Rueck

20. YES; it took a considerable amount of time, effort, and expense, but we feel that the excellent programs that were entered made it well worth it. We are most pleased to share TWO of the many excellent entries with you.

- ANSWERS TO SELF-TEST QUESTIONS FOR CHAPTER 3 -

1. Video memory - Parallel - 7 bits using video MEM.
2. Approximately 2500 cycles. Some metropolitan phone companies offer wideband coax, but it is very costly.
3. UART or software - UART or software.
4. Usually a UART since it can work up to 20K Baud, plus. However for SLOW Baud rates equal to or less than 110 Baud, the software approach is cheaper.
5. Program SPEED for handling variables B, C, and D.
6. To give you a CHANCE to get ahead of the line printer.
7. YOU - then the line printer.
8. Add line 35: IF B = 16383 GOTO 10.
9. Raises a FULL flag by shifting its flag flip-flip.
10. Because it is 16 bits wide. Therefore holds up to 65535.
11. Shift-right arrow returns 25 via CALL 02BH and shift-left arrows returns 24 via CaLL 02BH.
12. CALL 032AH sends 'A' to the line printer when 409CH = +1 and CALL 03BH ALWAYS sends 'A' to the line printer.
13. Yes - only one 8 bit byte needed to store any of them.
14. Yes - but wasteful as it is 16 bits wide.
15. 1500+.
16. Some of them YES, but be CAREFUL; stack, string addr, etc.
17. Silicon Valley and Exxon Enterprises' corp. headquarters. Dr. Federico Faggin, the Albert Einstein AND George Washington of microprocessors and microcomputers.
18. BURST modulation and JTIDS - NOTHING is totally secure.
19. International Telephone & Telegraph Corporation and their Chief Scientist, Dr. Henri Busignies.
20. Never-Never Land - "yes", Peter Pan said.

- ANSWERS TO SELF TEST QUESTIONS FOR CHAPTER 4 -

1. Pin 21 = maskable interrupt and pins 37 or 39 = ground.
2. To avoid having you cobble up your printed circuit boards.
3. Stores the ENABLE/DISABLE interrupt status.
4. Stores IFF1's status during a NON-MASKABLE interrupt, as the LAST thing we want is to have a maskable interrupt occur during non-maskable interrupt servicing; i.e., one of the first jobs the Z-80 does upon receipt of a maskable interrupt is to put IFF1 flip-flop in DISABLE status.
5. To sense a catastrophic power failure and set in motion a switch over from normal to battery backup power BEFORE volatile memory is wiped out.
6. No; it waits for one more instruction to be executed to avoid another INTERRUPT disrupting the RETURN from interrupt routine. Those Fagginsylvanians are SMART.
7. The stack.
8. IM2 - No - Have to cobble up the expansion interface.
9. Sure it can. See figure 4-4 on page 4-7.
10. Eight + eight = 16 (for counting in hex).
11. This is a 'GOTCHA' - The address is at 40DF & 40E0 hex.
12. No, but in NON-DISK is fairly reliable.
13. Because we don't know any better and became gun-shy.
14. The LDIR instruction.
15. TIME.
16. Just in case they are used by BASIC (which is highly unlikely). The IY register is ONLY used by disk BASIC.
17. a) The 'sort of' real time clock. b) To provide a 'fun' subject for a Chapter in Volume 3.
18. Yes.
19. Yes, BUT only SOMETIMES.
20. Mr. Grid Gridley, hisself. You do GOOD work, Grid.

- ANSWERS TO SELF TEST QUESTIONS FOR CHAPTER 5 -

1. LM 3900 Norton quad opamp. It 'conditions' (shapes) the the incoming cassette signal during CLOAD.

2. A 74LS175 quad 'D' flip-flop that accomplishes the following: a) data latch for cassette motor control b) data latch for MODESEL 32 or 64 characters per line c) modulator for cassette signal during CSAVE (brilliant).

3. Port 255 decoder for INP(255) or OUT255,xx.

4. Its input to pins 4 and 9 is derived from the Z-80 which determines whether it is a port INPUT or OUTPUT.

5. It acts as a switch between the conditioned cassette input signal to its pin 12 and output to data bus D7. It is controlled by the INSIG signal from Z25.

6. Use a 500 to 2000 cycle, 1 volt peak to peak signal from an LM555 to the cassette output (EAR) line to the TRS-80.

7. 256 of course, BUT we would need a real STACK of buffers and decoders/demultiplexers to accomplish it.

8. ABSOLUTELY NOT, never, no-way. UNLESS you are a highly skilled printed circuit board WORKER (with minutely small traces), do yourself a favor and DO NOT open up the key- board OR expansion interface. DISASTERVILLE, beckons.

9. Because it is a QUALITY piece of equipment, NOT crowded, WELL built, EASY to use, and worked 2 years + WITHOUT failure. ALSO, has 'TEST' feature and 110/220 vac input.

10. TWO - DBO zero & DBO one - YES, two.

11. Double-pole-single-throw - 120 volts ac at 3 amps.

12. Eight - eight.

13. Two - about 20 volts. You may swap 2000 volt units.

14. One - NO, does NOT load down the address bus which was a problem on early model TRS-80s.

15. 128

16. EASILY; see bottom of page 5-16.

17. Victor A. Rizzardi - designer of the VAR/80.

18. Rotate accumulator right circular (not THROUGH carry).

19. YES; up to 256 - see bottom of page 5-16.

20. See center page 5-28 - NO, but it will work - NO! NO! NO!

- ANSWERS TO SELF TEST QUESTIONS FOR CHAPTER 6 -

1. Epsco's DATRAC - $8000 each - 150 pounds - tube type.
2. Amount of time (seconds) for a voltage to rise to 63.2 percent of its original THROUGH a resistance R (ohms) when charging a capacitor C (farads). T = R times C.
3. John Napier - Scotland - Around 1610 AD !
4. Single slope, integrating-indirect type.
5. Gives the capacitor 'C' time to charge up to the NEARLY total source voltage.
6. Non-linearity - Offset - Scale factor.
7. The 'FLAKEY' electrolytic capacitor - Polarization memory (all those little molecular dipoles remember).
8. SLOW and EASILY fooled by 'noise'.
9. FLASH (parallel) type.
10. VERY expensive for us non-NASA or non-DOD types.
11. a) Successive approximation - feedback type.
    b) National Semiconductor for the ADC0808.
    c) Approximately $30 - $50 depending on source.
12. a) 8 channel multiplexer - 3 bit binary address decoder for selecting 8 inputs
    b) 8 bit output buffer/latch.
    c) End of conversion (EOC) FLAG output.
13. One difference; 4 bit binary address decoder = 16 inputs.
14. Mention the "Disassembled Handbook" when ordering.
15. NOT unless you install a voltage divider. Input impedance is > 1 megohm.....much like a vacuum tube voltmeter.
16. You MUST connect it directly to the keyboard so that the INTERRUPT function will operate properly.
17. NO - they serve only as reminders when typing the program.
18. Alpha Product's EXPANDABUS ADAPTOR.
19. JP 1A19H will OFTEN foul-up your resident BASIC program.
20. You get what you pay for. Its versatility and most IMPORTANTLY its quality are worth the extra dollars.

- ANSWERS TO SELF TEST QUESTIONS FOR CHAPTER 7 -

1. On a scale of 1 (good) to 10 (bad), electrolytics rate over 100, EXCEPT for price.

2. YES; but you MUST be able to read tea leaves too.

3. To depolarize the electrolytic capacitor - YES.

4. Beckman Instruments - 1967.

5. Parallel.

6. As a 'weighted current source' for D/A converters.

7. a) Simplicity.
   b) High speed.
   c) Modest cost.

8. Noise immunity.

9. About $8.40, but some are discontinued in the new 1981 Radio Shack catalog. Digi-Key carries them all.

10. NO; $3.37 from Digi-Key.

11. Yes, but even a cheapy volt-ohmmeter is better than nothing at all.

12. WRONG heat range and/or tip soldering pencil. Please re-read paragraph #5, page 7-11 if you are a newcomer.

13. Building the power supplies - No, only an hour or two at best AFTER you receive the parts.

14. YES; 12 vac RMS = peak voltage of 16.8 volts which is adequate for small loads.

15. It sets the amplification factor of the LF351 opamp, and hence the absolute (not relative) accuracy of conversion.

16. Not that we know of.

17. Fith, sith, seveth, and aith.

18. a) Faster. b) No auxiliary opamp required.

19. Yes; $3.87 = 50 cents more costly than the DAC0808.

20. See bottom of page 7-22. - $3.95 postpaid bargain.

CHAPTER 8
NO SELF TEST QUESTIONS AS THE PROGRAM IS PRESENTED FOR INFORMATION ONLY

- ANSWERS TO SELF TEST QUESTIONS FOR CHAPTER 9 -

1. NO; if NOT for money - YES; very much so UNLESS you already have PAID for it; i.e., Level II ROM.
2. YES - We think it is good, if free. - See Chapter 10.
3. Expansion interface, RS-232C, and MODEM or possibly a Peripheral People Interface for direct phone connection.
4. Yes and No. - Program speed makes it marginal without the Mumford Micro clock speed-up.
5. No; totally unnecessary and hard to get to, besides.
6. Electronic Industry Association members - EIA, 2001 "I" Street NW, Washington, DC 20006.
7. YES.
8. Forum 80 program, Peripheral People interface, 3 or 4 disks (or 2 double density), & phone answering circuit.
9. 232 - yes - no
10. 233 - 055H - 022H
11. 234
12. 234
13. 235 - 235
14. 165 - 229
15. Around 5 seconds or the bulletin board will automatically hang-up.
16. 'F' for full duplex and 'O' for originate.
17. Fold a piece of paper lengthwise and place over MODEM with swtiches in the 'O' and 'TST' (test) positions. Now RUN program.
18. Donald Stoner-W6TNS, Chairman of Peripheral People.
19. Sets 'Z' flag. - Sure it could and more LOGICALLY.
20. NO - Two Z-80s - Actually 2 microprocessors with independent ancillary circuits and memory.

- ANSWERS TO SELF TEST QUESTIONS FOR CHAPTER 10 -

1. General (then Captain) Squires, U.S. Army Signal Corps.
2. Charles Krum - Jay Morton (as in salt).

3. Ma Bell, known as AT&T (the world's BEST phone system).
4. Too argumentative to answer. No fist fights, please.

5. Submariner, pilot, author, editor, AND founder of: 73, BYTE, KILOBAUD, and 80 MICROCOMPUTING Magazines.
6. 50 - 54 MHz and 144 - 148 MHz.

7. 1/2 duplex has NO answer back echo AND no parity check.
8. Uncrowded (6M)-allows technician licensees-line of sight.

9. Least significant bit (LSB).
10. Filter OR phasing single sideband generating system.

11. Yes or No.......mostly yes, BUT the Model 3? A beauty!
12. Good quality coax, antenna 70' away, and a good ground.

13. Hamfest (lots in U.S.) or QST magazine 'Ham Ads'.
14. Hamtronics - see text for address.

15. Amateur Electronic Supply, Milwaukee, Wisconsin.
16. Obfuscation at its utmost. Look it UP, Gridley.

17. Ugandan Minister of Telecommunications (Idi Amin protege).
18. TWO - 7 bits.

19. YES - 300 Baud is very marginal....do not trust it.
20. T=TRASMIT, beginning address, and number of bytes to move.

21. R=RECEIVE, address to stash, and number of bytes to stash.
22. We (humbly) think not, but someone will.

23. We sure hope so and have tried our very BEST.
24. We sure hope so and have tried our very BEST. - NO !

25. The printed circuit board, layout, and superb components.
26. Much, much better.

27. Machine language speed - YES, now on order.
28. YES....if we have space.

29. YES....see page 10-36.
30. Come on, Gridley. We had hoped it would be a SURPRISE.

31. NO, it is about 50+ pages longer than planned. Win some ?
32. Because our printing costs increased 33 1/3 percent.

33. No promises, but hopefully summer '81 - About $22/copy.
34. YES, Gridley. SEND NO MONEY JUST A RESERVATION at $18. IF you send a deposit, will be returned & cost you $22.

- INDEX FOR VOLUMES 1, 2, & 3 -

- INDEX CONTINUED -

- INDEX CONTINUED -

- INDEX CONTINUED -

- APPENDIX B -

DISK PROGRAMS FROM VOLUMES 1, 2, & 3:

Richcraft has received a number of inquiries from computer science teachers and individuals regarding availability of the copyrighted programs in Volumes 1, 2, & 3. As such, we prevailed upon a professional duplicator to make them available. They are available to BOTH individuals and schools at $20 per 35 track double-sided disk, or $27 for 2 single-sided disks, postpaid. Programs included on the first disk are:

- DISK NO. 1 -

PROGRAM COPYRIGHT (c) 1980		FILE NAME:
Basic Functions Name List/BAS		A
Basic Functions Call Adresses/BAS		B
Integer Arithmetic	- Source Code w/comments	C
Integer Arithmetic	- Object Code	D
Single Precision Arithmetic	- Source Code w/comments	E
Single Precision Arithmetic	- Object Code	F
Double Precision Arithmetic	- Source Code w/comments	G
Double Precision Arithmetic	- Object Code	H
ROM Function Demonstration	- Source Code w/comments	I
ROM Function Demonstration	- Object Code	J
Multi-Base Number Conversion/BAS		K
LPRINT All Zeros With Slash	- Source Code w/comments	L
LPRINT All Zeros With Slash	- Object Code	M
Single Precision Decoding/BAS		N
Single Precision Decoding (no exponent sign)/BAS		O
Double Precision Decoding/BAS		P
Double Precision Decoding (no exponent sign)/BAS		Q
'JKL' LPRINT Out Video (123)	- Source Code w/comments	R
'JKL' LPRINT Out Video (123)	- Object Code	S
Storing Video In MEM	- Source Code w/comments	T
Storing Video In MEM	- Object Code	U
Split Screen Video	- Source Code w/comments	V
Split Screen Video	- Object Code	W
W4UCH Morse Code Transmit & Receive/BAS - Part 1		X
W4UCH Morse Code Transmit & Receive/BAS - Part 2		Y
TV Satellite Locator/BAS		Z

- APPENDIX B CONTINUED -

DISK PROGRAMS FROM VOLUME 3:

- DISK NO. 2 -

PROGRAM COPYRIGHT (c) 1981	FILE NAME:
Weird 3 - BASIC Disassembler from Chapter 1	A
Contest/BAS - 1st prize winning disassembler program	B
Fastone/BAS - MOST UNIQUE prize winning program	C
LPRINT whiLe 'A' register inputs to FIFO - source	D
LPRINT while 'A' register inputs to FIFO - object	E
Interrupt demo 2 - source from Chapter 4	F
Interrupt demo 2 - object	G
Interrupt demo 4 - source	H
Interrupt demo 4 - object	I
VAR/80 input demo - source from Chapter 5	J
VAR/80 input demo - object	K
Hybrid Analog 80 - source from Chapter 6	L
Hybrid Analog 80 - object	M
Digital to analog test/BAS from Chapter 7	N
Bulletin board demo - source from Chapter 9	O
Bulletin board demo - object	P
ASCII VHF data link - source from Chapter 10	Q
ASCII VHF data link - object	R

- VOLUME 4 RESERVATION - VOLUME 4 RESERVATION -

CUT ON DOTTED LINE AND MAIL TO RICHCRAFT

- -

Richcraft Engineering Ltd. - Drawer 1065
Wahmeda Industrial Park
Chautauqua, New York 14722

YES: reserve a copy of Volume 4, 'Disassembled Handbook For TRS-80' at $18 postpaid. WHEN ready (summer/fall '81), bill me and upon receipt of my check, please ship to:

NAME: ______________________________
STREET: ______________________________
CITY: ____________________ STATE: ______ ZIP: ________

NOTE: Volume 4 will probably list at $22. Your reservation guarantees $18 postpaid. SEND NO MONEY - SEND NO MONEY.